
ARTSOFT
MACH
ArtSoft
DIVISION OF NEWFANGLED SOLUTIONS

Mach4 Core API

Version 1.01

Authors: Brian Barker, Steve Murphree

Rev:1.01
2015-03-19

Contents

Getting Started

Supported Platforms

At this time the only supported platforms are Windows, Linux, Mac. The only plugins that will be suported for both Windows, Linux and Mac will need to be programmed in C++ or C.

Windows:
Plugins can be writen in any language that supports interfacing to a flat C API interface. Tests have been done with C++, C# and VB. Other languges such as Action Script, Delphi, Python and etc could be supported but are not tested due to the limited use in the windows world.

Linux:
GUIs need to be written in C or C++ with wxWidgets if they are to have a toolpath display. No testing has been conducted to see if other languages (wyPython, wxPerl...) with wxWigets will work. Any language with wx should work but it is out of the scope of this help manual to assist with that process.

Mac:
More work needs to be done on this platform :)

Compiling

Include the MachAPI.h file in your source files.

For the Windows Operationg System:

The dynamic C runtime should be used. /MD and /MDd for Visual Studio.

There are a few environment variables that can be used:

- **MACHSDK** should point to the Mach SDK folder. This environment variable can then be used in your projects to point to the inlude and lib directories in your project. e.g $(MACHSDK)\include and $(MACHSDK)\lib.
- **M4HOBBY** or should point to the Mach installation directory. Plugin projects can then use this environment variable to copy the resulting plugin the Plugins folder. e.g. In the post build event, add copy "$(TargetPath)" "$(M4HOBBY)Plugins"

Several compile time definitions are required or can be useful:

- Define **MACH_STATIC** in your project or define **MACH_STATIC** in your code before the MachAPI.h file is included. **This is required!**
- **_ATL_XP_TARGETING**: If you are using VS2013, it will not compile an executable that runs on windows XP by default. Defining **_ATL_XP_TARGETING** will correct this in conjuction with setting the linker minimum version to 5.01.
- **FAST_DEBUG**: A link library (MachAPIfd.lib) that uses the non debug version of the CRT (/MD) is provided for speed. Your code should then also be compiled with /MD and define FAST_DEBUG but you should turn on symbol information and link with debugging info. This keeps the debug CRT heap walking/memory checking routines from slowing things down in the debugger. Of course, it is a good idea to check you project with full debug settings periodically to check for memory leaks and such.
- **_CRT_SECURE_NO_WARNINGS**:, if you use a lot of the old C style functions that are considered as insecure, this will reduce your warning messages.

Linking

For the Windows Operationg System:

- For a Relase target, add MachAPI.lib to your library list.
- For a FastDebug target, add MachAPIfd.lib to your library list.
- For a Debug target, add MachAPId.lib to your library list.

API Reference by Category

The API documentation has been devided up into categories to help quickly locate API information for a given programming task.

- Plugins and Devices: API functions that are primarily used to create plugins with devices.
- Mach I/O: API functions that are primarily used to interface with I/O and to create I/O plugins.
- Mach Signals: API functions that work with Mach Signals.
- Mach Registers: API functions that work with Mach Registers.
- Motors: API functions that work with Motors.
- Screw Mapping: API functions that work with Motor screw maps.
- Axes: API functions that work with Axes.
- Motion: API functions that are primarily used to create motion plugins.
- Operation: API functions that are used to operate the machine (Cycle start, Feed Hold, etc...).
- GUI: API functions that are specific to GUI programming.
- Profile (INI) Settings: API functions for profile INI manipulation.
- General: General API functions.
- Gcode File: API functions to manipulate Gcode files.
- Tool Offsets: API functions dealing with tool offsets and tool selection.
- Jogging: Jogging API functions.
- MPGs: API functions for working with MPGs.
- Soft Limits: API functions for working with Soft Limits.
- Spindle: Spindle API functions.
- Scripting: API functions used for manipulating scripts.
- Status Messages: API functions for reporting error and status messages.
- License: API functions for working with license files.

Plugins and Devices

The Mach core uses loadable modules know as "plugins" that provide interfaces to devices and/or provides services. When the core is initialized, the Plugins directory searched and each plugin found is loaded with the following sequence:

1. The mcPluginLoad() entry point is called. The plugin should register itself with the core and provide its capabilites at this time.
2. The mcPluginInit() entry point is called for each Core instance. Devices should be registered at this time as well as any I/O and registers that belong to the device.

- mcDeviceGetHandle
- mcDeviceGetInfo
- mcDeviceGetInfoStruct
- mcDeviceGetNextHandle
- mcDeviceRegister
- mcPluginConfigure
- mcPluginCoreLoad
- mcPluginCoreUnload
- mcPluginDiagnostic
- mcPluginEnable
- mcPluginGetEnabled
- mcPluginGetInfoStruct
- mcPluginGetLicenseFeature
- mcPluginGetNextHandle
- mcPluginGetValid
- mcPluginInstall
- mcPluginRegister
- mcPluginRemove

mcDeviceGetHandle

C/C++ Syntax:

```
int mcDeviceGetHandle(
  MINSTANCE mInst,
  int devid,
  HMCDEV *hDev);
```

LUA Syntax:

```
hDev, rc = mc.mcDeviceGetHandle(
  number mInst,
  number devid)
```

Description: Retrieve the hDev handle specified by devid.

Parameters:

Parameter	Description
mInst	The controller instance.
devid	The device ID as it was registered in the core.
hDev	The address of a HMCDEV handle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_DEVICE_NOT_FOUND	No device with the ID specified by **devid**was found.

Notes:
None.

Usage:

```
int mInst = 0;
int devid = 0;
HMCDEV hDev;
//See if we can find a device with an I of zero
int rc = mcDeviceGetHandle(mInst, devid, &hDev;);
```

```
if (rc == MERROR_NOERROR) {
 // We found it!
}
```

mcDeviceGetInfo

C/C++ Syntax:

```
mcDeviceGetInfo(
  HMCDEV hDev,
  char *nameBuf,
  size_t nameBuflen,
  char *descBuf,
  size_t descBuflen,
  int *type,
  int *id);
```

LUA Syntax:

```
nameBuf, descBuf, type, id, rc = mc.mcDeviceGetInfo(HMCDEV hDev}
```

Description: Return infromation about a device.

Parameters:

Parameter	Description
hDev	The handle for the device.
nameBuf	The address of a string buffer to receive the device name.
nameBuflen	The nameBuf buffer length.
descBuf	The address of a string buffer to receive the device description.
descBuflen	The descBuf buffer length.
type	The address of an integer to receive the device type.
id	The address of an integer to receive the device ID.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
No Error.	
MERROR_INVALID_ARG	The **hDev** parameter was 0 or no device with that handle exists.

Notes:
None.

Usage:

```
int mInst=0;
HMCSIG hDev = 0;
devinfo_t devinf;
char nameBuf[80];
char descBuf[80];
int type = 0;
int id = 0;
//Look for all the IO devices and get their devinfo_t struct
while (mcDeviceGetNextHandle(mInst, DEV_TYPE_IO, hDev, &hDev;) == MERROR_NOERROR)
 if (hSig != 0) {
  mcDeviceGetInfo(hDev, nameBuf, sizeof(nameBuf),
    descBuf, sizeof(descBuf), &type;, &id;);
// do something with the device info.
 } else {
  break;
 }
}
```

mcDeviceGetInfoStruct

C/C++ Syntax:

```
int mcDeviceGetInfoStruct(
  HMCSIG hDev,
  devinfo_t *devinf);
```

LUA Syntax:

```
N/A
```

Description: Return infromation about a device.

Parameters:

Parameter	Description
hDev	The handle for the device.
devinf	The address of a devinfo struct to receive the device information.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The **hDev** parameter was 0.

Notes:

```
struct devinfo {
 char devName[80];
 char devDesc[80];
 int  devType;
 int  devId;
};
typedef struct devinfo devinfo_t;
```

Usage:

```
int mInst=0;
HMCSIG hDev = 0;
devinfo_t devinf;

//Look for all the IO devices and get their devinfo_t struct
```

```
while (mcDeviceGetNextHandle(mInst, DEV_TYPE_IO, hDev, &hDev;) == MERROR_NOERROR)
 if (hSig != 0) {
  mcDeviceGetInfoStruct(hDev, devinf);
// Do something with the device info.
 } else {
  break;
 }
}
```

mcDeviceGetNextHandle

C/C++ Syntax:

```
int mcDeviceGetNextHandle(
  MINSTANCE mInst,
  int devtype,
  HMCDEV startDev,
  HMCDEV *hDev);
```

LUA Syntax:

```
hDev, rc = mc.mcDeviceGetNextHandle(
  number mInst,
  number devtype,
  number startDev)
```

Description: Provides a means of looping through the available devices.

Parameters:

Parameter	Description
mInst	The controller instance.
devtype	Device type DEV_TYPE_NONE, DEV_TYPE_MOTION, DEV_TYPE_IO, DEV_TYPE_SCRIPT, DEV_TYPE_CONFIG, DEV_TYPE_DIAG.
startDev	A device handle defining the device from which to start.
hDev	A pointer to a HMCDEV to receive the next device handle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NODATA	No more device handles are available.

Notes:
None.

Usage:

```
int mInst=0;
HMCDEV hDev;
//Look for all the IO devices
while (mcDeviceGetNextHandle(mInst, DEV_TYPE_IO, hDev, &hDev;) == MERROR_NOERROR)
 if (hSig != 0) {
  // Do something with hDev.
 } else {
  break;
 }
}
```

mcDeviceRegister

C/C++ Syntax:

```
mcDeviceRegister(
 MINSTANCE mInst,
 HMCPLUG plugid,
 const char *DeviceName,
 const char *DeviceDesc,
 int Type,
 HMCDEV *hDev);
```

LUA Syntax:

```
N/A
```

Description: Register the device in the core.

Parameters:

Parameter	Description
mInst	The controller instance.
plugid	The ID of the plugin passed by mcPluginInit.
DeviceName	Name the device.
DeviceDesc	Description of the device.
Type	DEV_TYPE_NONE, DEV_TYPE_MOTION, DEV_TYPE_IO, DEV_TYPE_SCRIPT, DEV_TYPE_CONFIG, DEV_TYPE_DIAG, DEV_TYPE_REG, DEV_TYPE_PROBE2, DEV_TYPE_THREAD2, DEV_TYPE_RTAP, DEV_TYPE_RTAP2, DEV_TYPE_PROBE, DEV_TYPE_THREAD
hDev	The address of a HMCDEV to receive the returned device handle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

MERROR_PLUGIN_NOT_FOUND	The plugin referenced by **pluginid** was not found.
MERROR_NOT_CREATED	The device was not registered!

Notes:
Registers a device into the core and defines its capabilities.

Usage:

```
simControl::simControl(MINSTANCE mInst, HMCPLUG id)
{
 m_cid = mInst;
 m_id = id;
 m_timer = new simTimer(this);
 m_cycletime = .001;

 mcDeviceRegister(m_cid, m_id, "Sim0", "Simulation Device", DEV_TYPE_MOTION | DE

 mcRegisterIo(m_hDev, "Input0", "Input0", IO_TYPE_INPUT, &m;_Input0);
 mcRegisterIo(m_hDev, "Output0", "Output0", IO_TYPE_OUTPUT, &m;_Output0);

 if (!m_timer->Start(10, wxTIMER_ONE_SHOT)) {
  wxMessageBox(wxT("Timer could not start!"), wxT("Timer"));
 }
}
```

mcPluginConfigure

C/C++ Syntax:

```
int mcPluginConfigure(
  MINSTANCE mInst,
  int plugId);
```

LUA Syntax:

```
N/A
```

Description: Display the plugin configuration dialog for the given plugin ID.

Parameters:

Parameter	Description
mInst	The controller instance.
plugId	An integer specifying the plugin ID.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_PLUGIN_NOT_FOUND	The plugin specified by **plugId** was not found.

Notes:
Plugins can contain multiple devices. However, only one configuration dialog is used so plugins must have a dialog that is capable of configuring every device that belongs to it (if needed).

Usage:

```
MINSTANCE mInst = 0;
HMCPLUG hPlug = 0;
pluginfo_t pluginf;

// Loop through all of the plugins looking for plugins
// that have configuration dialogs.
while (mcPluginGetNextHandle(PLUG_TYPE_CONFIG, hPlug, &hPlug;) == MERROR_NOERROR)
 // Get the plugin info that contains the plugin ID.
```

```
 rc = mcPluginGetInfoStruct(hPlug, &pluginf;);
 if (rc == MERROR_NOERROR) {
  int pluginId = pluginf.plugId;
  // Call the configuration dialog.
  rc = mcPluginConfigure(mInst, pluginId);;
 }
}
```

mcPluginCoreLoad

C/C++ Syntax:

```
int mcPluginCoreLoad(
  const char *shortName);
```

LUA Syntax:

```
N/A
```

Description: Load a plugin into the core.

Parameters:

Parameter	Description
shortName	A string buffer specifying the base name of the plugin.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_PLUGIN_NOT_FOUND	The plugin specified by **shortName** was not found.

Notes:
Normally, all plugins are found and loaded by the core at stratup. This function provides a means to load a plugin after startup. For instance, if the plugin was installed after startup. The function uses the base name of the plugin to locate it by appending the platform specific extension (**m4pw** for Windows, **m4pl** for Linux, and **m4pm** for Mac).

Usage:

```
// Load the mcGalil plugin.
int rc = mcPluginCoreLoad("mcGalil");
```

mcPluginCoreUnload

C/C++ Syntax:

```
int mcPluginCoreUnload(
  const char *shortName);
```

LUA Syntax:

```
N/A
```

Description: Unload a plugin from the core.

Parameters:

Parameter	Description
shortName	A string buffer specifying the base name of the plugin.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_PLUGIN_NOT_FOUND	The plugin specified by **shortName** was not found.

Notes:
About the only reason to unload a plugin from the core is if it is being uninstalled. Plugins should be tested during their development to ensure that they can be unloaded safely without causing crashes or other abnormal behavior.

Usage:

```
// Unload the mcGalil plugin.
int rc = mcPluginCoreUnload("mcGalil");
```

mcPluginDiagnostic

C/C++ Syntax:

```
int mcPluginDiagnostic(
  MINSTANCE mInst,
  int pluginId);
```

LUA Syntax:

```
N/A
```

Description: Display the plugin diagnostic dialog for the given plugin ID.

Parameters:

Parameter	Description
mInst	The controller instance.
plugId	An integer specifying the plugin ID.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_PLUGIN_NOT_FOUND	The plugin specified by **plugId** was not found.

Notes:
Plugins can contain multiple devices. However, only one diagnostic dialog is used so plugins must have a dialog that is capable of diagnosing every device that belongs to it (if needed).

Usage:

```
MINSTANCE mInst = 0;
HMCPLUG hPlug = 0;
pluginfo_t pluginf;

// Loop through all of the plugins looking for plugins
// that have diagnostic dialogs.
while (mcPluginGetNextHandle(PLUG_TYPE_DIAG, hPlug, &hPlug;) == MERROR_NOERROR)
 // Get the plugin info that contains the plugin ID.
```

```
 rc = mcPluginGetInfoStruct(hPlug, &pluginf;);
 if (rc == MERROR_NOERROR) {
  int pluginId = pluginf.plugId;
  // Call the diagnostics dialog.
  rc = mcPluginDiagnostic(mInst, pluginId);;
 }
}
```

mcPluginEnable

C/C++ Syntax:

```
int mcPluginEnable(
  HMCPLUG hPlug,
  BOOL enable);
```

LUA Syntax:

```
N/A
```

Description: Enable a plugin.

Parameters:

Parameter	Description
hPlug	A HMCPLUG handle specifying the plugin.
enable	A BOOL specifying the plugin enable state.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_PLUGIN_NOT_FOUND	The plugin specified by **hPlug** was not found.

Notes:
None.

Usage:

```
HMCPLUG hPlug = 0;
// Loop through all of the plugins looking for plugins
// that have configuration dialogs.
while (mcPluginGetNextHandle(PLUG_TYPE_CONFIG, hPlug, &hPlug;) == MERROR_NOERROR)
 // Enable the plugin.
 rc = mcPluginEnable(hPlug, TRUE);
}
```

mcPluginGetEnabled

C/C++ Syntax:

```
int mcPluginGetEnabled(
  HMCPLUG hPlug,
  BOOL *enabled);
```

LUA Syntax:

```
N/A
```

Description:

Parameters:

Parameter	Description
hPlug	A HMCPLUG handle specifying the plugin.
enabled	The address of a BOOL to receive plugin enable state.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_PLUGIN_NOT_FOUND	The plugin specified by **hPlug** was not found.
MERROR_INVALID_ARG	**enabled** is NULL.

Notes:

Usage:

```
HMCPLUG hPlug = 0;
BOOL enabled = FALSE;
// Loop through all of the plugins looking for plugins
// that have configuration dialogs.
while (mcPluginGetNextHandle(PLUG_TYPE_CONFIG, hPlug, &hPlug;) == MERROR_NOERROR)
 // See if the plugin is enabled..
 rc = mcPluginGetEnabled(hPlug, &enabled;);
 if (rc == MERROR_NOERROR) {
  if (enabled == TRUE) {
   // The plugin is enabled!!!
  } else {
   // The plugin is not enabled!!!
```

```
    }
  }
}
```

mcPluginGetInfoStruct

C/C++ Syntax:

```
mcPluginGetInfoStruct(HMCPLUG hPlug, pluginfo_t *pluginf);
```

LUA Syntax:

```
N/A
```

Description:

Parameters:

Parameter	Description
hPlug	A HMCPLUG handle specifying the plugin.
pluginf	The address of a pluginfo_t struct to receive the plugin information.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_PLUGIN_NOT_FOUND	The plugin specified by **hPlug** was not found.
MERROR_INVALID_ARG	**pluginf** is NULL.

Notes:

Usage:

```
HMCPLUG hPlug = 0;
pluginfo_t pluginf;

// Loop through all of the plugins.
while (mcPluginGetNextHandle(PLUG_TYPE_ALL, hPlug, &hPlug;) == MERROR_NOERROR) {
 // Get the plugin info that contains the plugin ID.
 rc = mcPluginGetInfoStruct(hPlug, &pluginf;);
 if (rc == MERROR_NOERROR) {
  int pluginId = pluginf.plugId;
  ...
 }
}
```

mcPluginGetLicenseFeature

C/C++ Syntax:

```
int mcPluginGetLicenseFeature(
  HMCPLUG hPlug,
  int index,
  char *buf,
  int bufSize,
  BOOL *status);
```

LUA Syntax:

```
N/A
```

Description: Get the plugin licensed features by the given index and return their status (licensed or not).

Parameters:

Parameter	Description
mInst	The controller instance.
index	An integer specifying the index of the license feature to retrieve.
buf	A string buffer to receive the license feature name.
bufSize	The length of the string buffer.
status	The address of a BOOL to receive the license status of the feature (TRUE/FALSE).

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	**hPlug** is zero, **buf** or **status** is NULL, **bufSize** is <= 0.
MERROR_PLUGIN_NOT_FOUND	The plugin specified by **hPlug** was not found.
MERROR_NODATA	No more licensed features can be found.

Notes:

Usage:

```
HMCPLUG hPlug = 0;
// Loop through all of the plugins.
while (mcPluginGetNextHandle(PLUG_TYPE_ALL, hPlug, &hPlug;) == MERROR_NOERROR) {
 // Get all of the plugin license features
 int index = 0;
 char buf[80];
 BOOL licensed = FALSE;
 while (mcPluginGetLicenseFeature(hPlug, index, buf, sizeof(buf), &licensed;) ==
  printf("license feature %s is %s.\n", buf, licensed == TRUE ? "licensed" : "not
  index++;
 }
}
```

mcPluginGetNextHandle

C/C++ Syntax:

```
int mcPluginGetNextHandle(
  int plugType,
  HMCPLUG startPlug,
  HMCPLUG *hPlug);
```

LUA Syntax:

```
N/A
```

Description: Provides a means of looping through all or only plugins with specific attributes.

Parameters:

Parameter	Description
plugType	An integer specifying the plugin attributes. (PLUG_TYPE_NONE, PLUG_TYPE_MOTION, PLUG_TYPE_IO, PLUG_TYPE_SCRIPT, PLUG_TYPE_CONFIG, PLUG_TYPE_DIAG, PLUG_TYPE_REG, PLUG_TYPE_JOG, PLUG_TYPE_ALL)
startPlug	The starting HMCPLUG handle with which to retrieve the next handle in the list. (Must be 0 to retrieve the first handle in the list)
hPlug	The address of a HMCPLUG to receive the next plugin handle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_NODATA	No more plugins with the specified attributes can be found.

Notes:

Usage:

```
HMCPLUG hPlug = 0;
```

```
// Loop through all of the plugins looking for plugins
// that have configuration dialogs and privide I/O.
int attr = PLUG_TYPE_CONFIG | PLUG_TYPE_IO;
while (mcPluginGetNextHandle(attr, hPlug, &hPlug;) == MERROR_NOERROR) {
 // Enable the plugin.
 rc = mcPluginEnable(hPlug, TRUE);
}
```

mcPluginGetValid

C/C++ Syntax:

```
int mcPluginGetValid(
  HMCPLUG hPlug,
  BOOL *valid);
```

LUA Syntax:

```
N/A
```

Description: Check to see if the plugin is valid.

Parameters:

Parameter	Description
hPlug	A HMCPLUG specifying the plugin handle.
valid	The address of a BOOL to receive the plugins validity status.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	**hPlug** is 0 or **valid** is NULL
MERROR_PLUGIN_NOT_FOUND	The plugin specified by **hPlug** cannot be found.

Notes:

Plugin validity is a function of the plugin signature resolving to a certified Mach developer. If no signature file is provided with the plugin or it is invalid, the plugin will not function.

Plugin signatures are generated with a developer key. Developer keys can be requested by contacting Newfangled Solutions, LLC.

Usage:

```
HMCPLUG hPlug = 0;
BOOL valid = FALSE;
// Loop through all of the plugins checking to see if they are valid.
```

```
while (mcPluginGetNextHandle(PLUG_TYPE_ALL, hPlug, &hPlug;) == MERROR_NOERROR) {
 // Check the plugin validity.
 rc = mcPluginGetValid(hPlug, &valid;);
 if (rc == MERROR_NOERROR) {
  if (valid == TRUE) {
   // The plugin is valid and will function.
  } else {
   // The plugin is invalid and will not function.
  }
 }
}
```

mcPluginInstall

C/C++ Syntax:

```
int mcPluginInstall(
  const char *m4plug);
```

LUA Syntax:

```
N/A
```

Description: Provides a means of installing a plugin from a plugin archive.

Parameters:

Parameter	Description
m4Plug	A string buffer specifying the plugin archive to install.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_PLUGIN_NOT_FOUND	The plugin archive specified by **m4Plug** is not valid.

Notes:
Plugin archive name extentions vary by the plafrom. **m4plugw** for Windows, **m4plugl** for Linux, **m4plugm** for Mac. If one were to try and install a windows plugin archive on a Linux machine, then MERROR_PLUGIN_NOT_FOUND would be returned.

Usage:

```
// Install the Windows mcGalil plugin from an installation archive.
int rc = mcPluginInstall("mcGalil.m4Plugw");
if (rc == MERROR_NOERROR) {
 //The plugin installed successfully.
}
```

mcPluginRegister

C/C++ Syntax:

```
int mcPluginRegister(
  HMCPLUG hPlug,
  const char *DeveloperId,
  const char *Desc,
  const char *Version,
  int Type);
```

LUA Syntax:

```
N/A
```

Description: Registers the plugin to the core.

Parameters:

Parameter	Description
hPlug	A HMCPLUG handle (generated and provided by the core) that defines the current plugin.
DeveloperId	A string buffer specifying the developer ID.
Desc	A string buffer specifying the plugins description.
Version	A string buffer specifying the plugins version.
Type	A integer specifying the plugins attributes (or capabilities). It can be the logical **OR** combination of PLUG_TYPE_MOTION, PLUG_TYPE_IO, PLUG_TYPE_SCRIPT, PLUG_TYPE_CONFIG, PLUG_TYPE_DIAG, PLUG_TYPE_REG, and PLUG_TYPE_JOG.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_PLUGIN_NOT_FOUND	The plugin specified by **hPlug** is not valid.

Notes:
Once a plugin is found by the core, the core loads the plugin and calls its mcPluginLoad() entry point

function. The plugin should then register itself with the core at this time.

Usage:

```
// This function gets called (only once) when the Plugin is loaded by Mach Core.
MCP_API int MCP_APIENTRY mcPluginLoad(HMCPLUG id)
{
 ...
 mcPluginRegister(id, "Newfangled",
  "Simulator - Newfangled Solutions", MC_VERSION_STR,
  PLUG_TYPE_MOTION | PLUG_TYPE_IO  | PLUG_TYPE_REG | PLUG_TYPE_CONFIG | PLUG_TYPE
 return(MERROR_NOERROR);
}
```

mcPluginRemove

C/C++ Syntax:

```
int mcPluginRemove(const char *shortName);
```

LUA Syntax:

```
N/A
```

Description: Unloads a plugin and then rmoves the plugin from the core's Plugins directory.

Parameters:

Parameter	Description
shortName	A string buffer specifying the short name (base name).

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
If the plugin sepcified by **shortName** was installed using a plugin archive, then all of the files installed by said archive are removed. Otherwise, only the plugin itself is removed from the Plugins directory.

Usage:

```
// Remove the mcGalil plugin.
MINSTANCE mInst = 0;
int rc = mcPluginRemove("mcGalil");
```

Mach I/O

I/O is registered by devices that are registered by plugins. The Mach core signals can then be mapped to the I/O. The mcIoGetHandle() function can be used to get a handle to an I/O object. It takes a path in the form of "**device/ioName**". Once a handle is obtained, the state can be checked or set depending if the I/O object is an input or an output. Scripts can operate with I/O directly or through the signal interface, whichever is the most useful for a given task.

mcIoGetHandle

C/C++ Syntax:

```
mcIoGetHandle(
 MINSTANCE mInst,
 const char *path,
 HMCIO *hIo);
```

LUA Syntax:

```
hIo, rc = mc.mcIoGetHandle(
  number mInst,
  string path)
```

Description: Retrieve the IO handle given a named path.

Parameters:

Parameter	Description
mInst	The controller instance.
path	A string representing the named path. e.g. devname/ioname
hIo	The address of a HMCIO handle

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_IO_NOT_FOUND	No IO handle available for the given path.

Notes:
None.

Usage:

```
HMCIO hIo;
char *path = "Sim0/Input0";

mcIoGetHandle(m_cid, path, &hIo;);
```

mcIoGetInfoStruct

C/C++ Syntax:

```
mcIoGetInfoStruct(
 HMCIO hIo,
 ioinfo_t *ioinf);
```

LUA Syntax:

```
N/A
```

Description: Retrieve the current state of the IO handle.

Parameters:

Parameter	Description
hIo	The IO handle.
ioinf	The address of an ioinf struct to receive the IO information.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The **hIo** handle is 0 or invalid, or the **ioinf** parameter is NULL

Notes:
The hIo parameter **MUST** be a valid IO handle. Otherwise, the function will crash.

```
struct ioinfo {
 char ioName[80];
 char ioDesc[80];
 int  ioType;
 HMCDEV ioDev;
 HMCSIG ioMappedSignals[MAX_MAPPED_SIGNAL];
 void *ioUserData;
 int ioInput;
};
typedef struct ioinfo ioinfo_t;
```

Usage:

```
HMCDEV hDev = 0;
devinfo_t devinf;
int rc;

// Get all IO information from every registered device.
while (mcDeviceGetNextHandle(0, DEV_TYPE_IO, hDev, &hDev;) == MERROR_NOERROR) {
 if (mcDeviceGetInfoStruct(hDev, &devinf;) == MERROR_NOERROR) {
  HMCIO hIo = 0;
  while (mcIoGetNextHandle(hDev, hIo, &hIo;) == MERROR_NOERROR) {
   ioinfo_t ioinf;
   rc = mcIoGetInfoStruct(hIo, &ioinf;);
   if (rc ==

MERROR_NOERROR No Error.) {
    // IO information successfuly retrieved.
   }
  }
 }
}
```

mcIoGetNextHandle

C/C++ Syntax:

```
mcIoGetNextHandle(
 HMCDEV hDev,
 HMCIO startIo)
 HMCIO *hIo)
```

Description: Provides a means to loop though the registered IO of a registered device.

Parameters:

Parameter	Description
hDev	The device handle.
startIo	The starting IO handle. (0 to begin)
hIo	The address to a HMCIO handle to receive the next IO handle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	The hDev paramete was 0.
MERROR_NODATA	No IO handle available.

Notes:
The hSig parameter **MUST** be a valid signal handle. Otherwise, the function will crash.
Useful in a GUI where retrieving information to display to the user is needed.

Usage:

```
HMCDEV hDev = 0;
devinfo_t devinf;
int rc;

// Get all IO information from every registered device.
while (mcDeviceGetNextHandle(0, DEV_TYPE_IO, hDev, &hDev;) ==

MERROR_NOERROR No Error.) {
 if (mcDeviceGetInfoStruct(hDev, &devinf;) ==
```

```
MERROR_NOERROR No Error.) {
  HMCIO hIo = 0;
  while (mcIoGetNextHandle(hDev, hIo, &hIo;) ==

MERROR_NOERROR No Error.) {
   ioinfo_t ioinf;
   rc = mcIoGetInfoStruct(hIo, &ioinf;);
   if (rc ==

MERROR_NOERROR No Error.) {
    // IO information successfuly retrieved.
   }
  }
 }
}
```

mcIoGetState

C/C++ Syntax:

```
mcIoGetState(
 HMCIO hIo,
 bool *state);
```

Description: Retrieve the current state of the IO handle.

Parameters:

Parameter	Description
hIo	The IO handle.
state	The address of a bool to receive the state status.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
No Error.	
MERROR_INVALID_ARG	The hIo parameter was 0.

Notes:
The hIo parameter **MUST** be a valid IO handle. Otherwise, the function will crash.

Usage:

```
simControl::simControl(MINSTANCE mInst, HMCPLUG id)
{
 m_cid = mInst;
 m_id = id;
 m_timer = new simTimer(this);
 m_cycletime = .001;
 bool oldstate = false;
 bool newstate = false;

 mcDeviceRegister(m_cid, m_id, "Sim0", "Simulation Device", DEV_TYPE_MOTION | DEV

 mcIoRegister(m_hDev, "Input0", "Input0", IO_TYPE_INPUT, &m;_Input0);
 mcIoGetState(m_Input0, &oldstate;);
 newstate = simGetIO(0);
 fi (newstate != oldstate) {
  mcIoSetState(m_Input0, newstate);
 }
```

}

mcIoGetType

C/C++ Syntax:

```
mcIoGetType(
 HMCIO hIo,
 int *type);
```

LUA Syntax:

```
type, rc = mcIoGetType(
 number hIo)
```

Description: Retrieve the type (input or output) the IO object.

Parameters:

Parameter	Description
hIo	The IO handle.
type	The address of an integer to receive the type information. (IO_TYPE_INPUT or IO_TYPE_OUTPUT)

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The **hIo** parameter was 0 or invalid.

Notes:
The hIo parameter **MUST** be a valid IO handle. Otherwise, the function will crash.

Usage:

```
simControl::simControl(MINSTANCE mInst, HMCPLUG id)
{
 m_cid = mInst;
 m_id = id;
 m_timer = new simTimer(this);
 m_cycletime = .001;
 bool oldstate = false;
 bool newstate = false;

 mcDeviceRegister(m_cid, m_id, "Sim0", "Simulation Device", DEV_TYPE_MOTION | DE
```

```
 mcIoRegister(m_hDev, "Input0", "Input0", IO_TYPE_INPUT, &m;_Input0);
 mcIoGetType(m_Input0, &type;);
// type will equal IO_TYPE_INPUT
}
```

mcIoGetUserData

C/C++ Syntax:

```
mcIoGetUserData(
 HMCIO hIo,
 void **data);
```

LUA Syntax:

```
N/A
```

Description: Retrieve the type user data pointer associated with the I/O object..

Parameters:

Parameter	Description
hIo	The IO handle.
data	The address of a pointer to receive the user data.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The **hIo** parameter was 0 or invalid.

Notes:
The hIo parameter **MUST** be a valid IO handle. Otherwise, the function will crash.

Usage:

```
struct myData {
 int myPortNumber;
 int myPinNumber;
};

void GetUserData(void)
{
 MINSTANCE mInst = 0;
 HMCIO hIo;
 void *data;
 if (mcIoGetHandle(mInst, "Sim0/Input0", &hIo;) == MERROR_NOERROR) {
  mcIoGetUserData(hIo, &data;);
// type cast the pointer
```

```
  struct myData *mData = (struct myData *)data;
 }
}
```

mcIoIsEnabled

C/C++ Syntax:

```
mcIoIsEnabled (
  HMCIO hIo,
  bool *enabled);
```

Description: Check to see if IO device has been enabled.

Parameters:

Parameter	Description
hSig	A sginal handle obtained from mcGetSignalHandle().
state	The address of a bool to receive the enabled state.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
No Error.	
MERROR_IO_NOT_FOUND	The IO parameter was not found.

Notes:
The hIo parameter **MUST** be a valid IO handle. Otherwise, the function will return MERROR_IO_NOT_FOUND.

Usage:

```
HMCIO hIo;
char *path = "Sim0/Input0";
bool enabled;
mcIoGetHandle(m_cid, path, &hIo;);//Get the handle
mcIoIsEnabled(hIo , &enabled;);//Get the enabled state of the IO
```

mcIoRegister

C/C++ Syntax:

```
mcIoRegister(
 HMCDEV hDev,
 const char *IoName,
 const char *IoDesc,
 int Type,
 HMCIO *hIo);
```

LUA Syntax:

```
N/A
```

Description: Register the I/O object in the Mach core.

Parameters:

Parameter	Description
hDev	Pointer to the device.
IoName	Name of the IO point.
IoDesc	Description of the IO.
Type	IO_TYPE_NONE, IO_TYPE_INPUT, IO_TYPE_OUTPUT, IO_TYPE_ANA_INPUT, IO_TYPE_ANA_OUTPUT.
hIo	Address of the IO point

Returns:

Return Code	Description
MERROR_NOERROR	No Error.

Notes:
The hDev parameter **MUST** be a valid device handle. Otherwise, the function will crash.

Usage:

```
simControl::simControl(MINSTANCE mInst, HMCPLUG id)
{
 m_cid = mInst;
 m_id = id;
 m_timer = new simTimer(this);
```

```
 m_cycletime = .001;

 mcDeviceRegister(m_cid, m_id, "Sim0", "Simulation Device", DEV_TYPE_MOTION | DEV

 mcIoRegister(m_hDev, "Input0", "Input0", IO_TYPE_INPUT, &m;_Input0);
 mcIoRegister(m_hDev, "Output0", "Output0", IO_TYPE_OUTPUT, &m;_Output0);

 if (!m_timer->Start(10, wxTIMER_ONE_SHOT)) {
  wxMessageBox(wxT("Timer could not start!"), wxT("Timer"));
 }
}
```

mcIoSetDesc

C/C++ Syntax:

```
mcIoSetDesc(
 HMCIO hIo,
 const char *desc);
```

LUA Syntax:

```
rc = mc.mcIoSetDesc(
 number hIo,
 string desc)
```

Description: Sets the decription of a registerd I/O object.

Parameters:

Parameter	Description
hIo	The I/O handle.
desc	The description to apply to the I/O object.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The hIo parameter was 0 or invalid.

Notes:
The hIo parameter **MUST** be a valid IO object handle. Otherwise, the function will crash.

Usage:

```
HMCIO hIo;
char *path = "Sim0/Input0";

if (mcIoGetHandle(m_cid, path, &hIo;) == MERROR_NOERROR) {
 mcIoSetDesc(hIo, "my mew description);
}
```

mcIoSetName

C/C++ Syntax:

```
mcIoSetName(
 HMCIO hIo,
 const char *name);
```

LUA Syntax:

```
rc = mc.mcIoSetName(
 number hIo,
 string name)
```

Description: Sets the name of a registerd I/O object.

Parameters:

Parameter	Description
hIo	The I/O handle.
name	The name to apply to the I/O object.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The hIo parameter was 0 or invalid.

Notes:
The hIo parameter **MUST** be a valid IO object handle. Otherwise, the function will crash.

Usage:

```
HMCIO hIo;
char *path = "Sim0/Input0";

if (mcIoGetHandle(m_cid, path, &hIo;) == MERROR_NOERROR) {
 mcIoSetName(hIo, "Limit0++");
// The name is changed from Input0 to Limit0++
}
```

mcIoSetState

C/C++ Syntax:

```
mcIoSetState(
 HMCIO hIo,
 bool state);
```

Description: Sets the state of a registerd I/O object.

Parameters:

Parameter	Description
hIo	The I/O object handle.
state	The state of I/O object

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The hIo parameter was 0.

Notes:
The hIo parameter **MUST** be a valid IO object handle. Otherwise, the function will crash.
A value of true means the I/O object is in a high state. This function should only be used upon a state change. It is a programming error otherwise and will cause messages to SPAM the core and GUI.

Usage:

```
simControl::simControl(MINSTANCE mInst, HMCPLUG id)
{
 m_cid = mInst;
 m_id = id;
 m_timer = new simTimer(this);
 m_cycletime = .001;
 bool oldstate = false;
 bool newstate = false;

 mcDeviceRegister(m_cid, m_id, "Sim0", "Simulation Device", DEV_TYPE_MOTION | DE

 mcIoRegister(m_hDev, "Input0", "Input0", IO_TYPE_INPUT, &m;_Input0);
 mcIoGetState(m_Input0, &oldstate;);
 newstate = simGetIO(0);
 if (newstate != oldstate) {
```

```
  mcIoSetState(m_Input0, newstate);
 }
}
```

mcIoSetType

C/C++ Syntax:

```
mcIoSetType(
 HMCIO hIo,
 int type);
```

LUA Syntax:

```
rc = mcIoSetType(
 number hIo
 number type)
```

Description: Retrieve the type (input or output) the IO object.

Parameters:

Parameter	Description
hIo	The I/O object handle.
type	An integer specifying the type of the I/O object. (IO_TYPE_INPUT or IO_TYPE_OUTPUT)

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The **hIo** parameter was 0 or invalid.

Notes:
The hIo parameter **MUST** be a valid IO handle. Otherwise, the function will crash.

Usage:

```
simControl::simControl(MINSTANCE mInst, HMCPLUG id)
{
 m_cid = mInst;
 m_id = id;
 m_timer = new simTimer(this);
 m_cycletime = .001;
 bool oldstate = false;
 bool newstate = false;

 mcDeviceRegister(m_cid, m_id, "Sim0", "Simulation Device", DEV_TYPE_MOTION | DE
```

```
 mcIoRegister(m_hDev, "Input0", "Input0", IO_TYPE_INPUT, &m;_Input0);
 mcIoSetType(m_Input0, IO_TYPE_OUTPUT);
// change the type to IO_TYPE_OUTPUT
}
```

mcIoSetUserData

C/C++ Syntax:

```
mcIoSetUserData(
 HMCIO hIo,
 void **data);
```

LUA Syntax:

```
N/A
```

Description: Retrieve the type user data pointer associated with the I/O object..

Parameters:

Parameter	Description
hIo	The IO handle.
data	The address of a pointer to receive the user data.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The **hIo** parameter was 0 or invalid.

Notes:
The hIo parameter **MUST** be a valid IO handle. Otherwise, the function will crash.

Usage:

```
struct myData {
 int myPortNumber;
 int myPinNumber;
};

struct myData data;

void GetUserData(void)
{
 MINSTANCE mInst = 0;
 HMCIO hIo;
 struct myData data;
 data.myPortNumber = 1;
```

```
 data.myPinNumber = 12;
 if (mcIoGetHandle(mInst, "Sim0/Input0", &hIo;) == MERROR_NOERROR) {
  mcIoSetUserData(hIo, &data;);
 }
}
```

mcIoSyncSignal

C/C++ Syntax:

```
mcIoSyncSignal(
 HMCIO hIo,
 BOOL state);
```

LUA Syntax:

```
N/A
```

Description: A special function that I/O plugins can use to force the synchronization of a Mach output signal.

Parameters:

Parameter	Description
hIo	The I/O object handle.
state	An BOOL specifying the state to sync to the I/O objects mapped core output signal

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The **hIo** parameter was 0 or invalid.

Notes:
The hIo parameter **MUST** be a valid IO handle. Otherwise, the function will crash. This function is primarily used to synchronize the core output signal when setting/clearing outputs are coordinated with machine motion. (M63/M63)

Usage:

```
N/A
```

mcIoUnregister

C/C++ Syntax:

```
mcIoUnregister(
 HMCDEV hDev,
 HMCIO hIo);
```

LUA Syntax:

```
N/A
```

Description: Unregister the I/O object in the Mach core.

Parameters:

Parameter	Description
hDev	Pointer to the device.
hIo	The /O object handle to remove from the core..

Returns:

Return Code	Description
MERROR_NOERROR	No Error.

Notes:
The hDev parameter **MUST** be a valid device handle. Otherwise, the function will crash.

Usage:

N/A

Mach Signals

The Mach core communicates with the outside world via a signal mechanism. All signals are "owned" by the core. The core drives output signals and input signals direct the core. This means that a user should never set an output signal and the core will never set an input signal.

It is important to note that signals are not I/O. However, output signals can drive outputs and inputs can drive input signals if they are mapped to repective I/O objects. This mapping is performed with mcSignalMap(). The I/O device has no need to know what signals to which any I/O may be mapped. This allows for abstraction of the signal functions.

Signals operations use a handle that represents the signal object in the core. This handle can be obtained via mcSignalGetHandle() by providing a signal ID. Signal IDs are defined in the MachAPI.h header file. Input signal ID definitions begin with **ISIG** and output signal ID definitions begin with **OSIG**.

The state of input and output signals can be retieved via mcSignalGetState(). The the state of input signals can be set via mcSignalSetState().

Signal information can be retireve via mcSignalGetInfo() and mcSignalGetInfoStruct().

Input Signals

Signal	Description
ISIG_PROBE	Probe
ISIG_PROBE1	Probe1
ISIG_PROBE2	Probe2
ISIG_PROBE3	Probe3
ISIG_INDEX	Index
ISIG_INPUT0	Input #0
ISIG_INPUT1	Input #1
ISIG_INPUT2	Input #2
ISIG_INPUT3	Input #3
ISIG_INPUT4	Input #4
ISIG_INPUT5	Input #5
ISIG_INPUT6	Input #6
ISIG_INPUT7	Input #7
ISIG_INPUT8	Input #8
ISIG_INPUT9	Input #9
ISIG_INPUT10	Input #10
ISIG_INPUT11	Input #11
ISIG_INPUT12	Input #12
ISIG_INPUT13	Input #13
ISIG_INPUT14	Input #14
ISIG_INPUT15	Input #15
ISIG_INPUT16	Input #16
ISIG_INPUT17	Input #17
ISIG_INPUT18	Input #18
ISIG_INPUT19	Input #19
ISIG_INPUT20	Input #20
ISIG_INPUT21	Input #21
ISIG_INPUT22	Input #22
ISIG_INPUT23	Input #23

ISIG_INPUT24	Input #24
ISIG_INPUT25	Input #25
ISIG_INPUT26	Input #26
ISIG_INPUT27	Input #27
ISIG_INPUT28	Input #28
ISIG_INPUT29	Input #29
ISIG_INPUT30	Input #30
ISIG_INPUT31	Input #31
ISIG_INPUT32	Input #32
ISIG_INPUT33	Input #33
ISIG_INPUT34	Input #34
ISIG_INPUT35	Input #35
ISIG_INPUT36	Input #36
ISIG_INPUT37	Input #37
ISIG_INPUT38	Input #38
ISIG_INPUT39	Input #39
ISIG_INPUT40	Input #40
ISIG_INPUT41	Input #41
ISIG_INPUT42	Input #42
ISIG_INPUT43	Input #43
ISIG_INPUT44	Input #44
ISIG_INPUT45	Input #45
ISIG_INPUT46	Input #46
ISIG_INPUT47	Input #47
ISIG_INPUT48	Input #48
ISIG_INPUT49	Input #49
ISIG_INPUT50	Input #50
ISIG_INPUT51	Input #51
ISIG_INPUT52	Input #52
ISIG_INPUT53	Input #53
ISIG_INPUT54	Input #54
ISIG_INPUT55	Input #55
ISIG_INPUT56	Input #56
ISIG_INPUT57	Input #57
ISIG_INPUT58	Input #58

ISIG_INPUT59	Input #59
ISIG_INPUT60	Input #60
ISIG_INPUT61	Input #61
ISIG_INPUT62	Input #62
ISIG_INPUT63	Input #63
ISIG_LIMITOVER	Limit Override
ISIG_EMERGENCY	E-Stop
ISIG_THCON	THC On
ISIG_THCUP	THC Up
ISIG_THCDOWN	THC Down
ISIG_TIMING	Timing
ISIG_JOGXP	Jog X+
ISIG_JOGXN	Jog X-
ISIG_JOGYP	Jog Y+
ISIG_JOGYN	Jog Y-
ISIG_JOGZP	Jog Z+
ISIG_JOGZN	Jog Z-
ISIG_JOGAP	Jog A+
ISIG_JOGAN	Jog A-
ISIG_JOGBP	Jog B+
ISIG_JOGBN	Jog B-
ISIG_JOGCP	Jog C+
ISIG_JOGCN	Jog C-
ISIG_SPINDLE_AT_SPEED	Spindle At Speed
ISIG_SPINDLE_AT_ZERO	Spindle At Zero
ISIG_MOTOR0_HOME	Motor 0 Home
ISIG_MOTOR0_PLUS	Motor 0 ++
ISIG_MOTOR0_MINUS	Motor 0 --
ISIG_MOTOR1_HOME	Motor 1 Home
ISIG_MOTOR1_PLUS	Motor 1 ++
ISIG_MOTOR1_MINUS	Motor 1 --
ISIG_MOTOR2_HOME	Motor 2 Home
ISIG_MOTOR2_PLUS	Motor 2 ++
ISIG_MOTOR2_MINUS	Motor 2 --

ISIG_MOTOR3_HOME	Motor 3 Home
ISIG_MOTOR3_PLUS	Motor 3 ++
ISIG_MOTOR3_MINUS	Motor 3 --
ISIG_MOTOR4_HOME	Motor 4 Home
ISIG_MOTOR4_PLUS	Motor 4 ++
ISIG_MOTOR4_MINUS	Motor 4 --
ISIG_MOTOR5_HOME	Motor 5 Home
ISIG_MOTOR5_PLUS	Motor 5 ++
ISIG_MOTOR5_MINUS	Motor 5 --
ISIG_MOTOR6_HOME	Motor 6 Home
ISIG_MOTOR6_PLUS	Motor 6 ++
ISIG_MOTOR6_MINUS	Motor 6 --
ISIG_MOTOR7_HOME	Motor 7 Home
ISIG_MOTOR7_PLUS	Motor 7 ++
ISIG_MOTOR7_MINUS	Motor 7 --
ISIG_MOTOR8_HOME	Motor 8 Home
ISIG_MOTOR8_PLUS	Motor 8 ++
ISIG_MOTOR8_MINUS	Motor 8 --
ISIG_MOTOR9_HOME	Motor 9 Home
ISIG_MOTOR9_PLUS	Motor 9 ++
ISIG_MOTOR9_MINUS	Motor 9 --
ISIG_MOTOR10_HOME	Motor 10 Home
ISIG_MOTOR10_PLUS	Motor 10 ++
ISIG_MOTOR10_MINUS	Motor 10 --
ISIG_MOTOR11_HOME	Motor 11 Home
ISIG_MOTOR11_PLUS	Motor 11 ++
ISIG_MOTOR11_MINUS	Motor 11 --
ISIG_MOTOR12_HOME	Motor 12 Home
ISIG_MOTOR12_PLUS	Motor 12 ++
ISIG_MOTOR12_MINUS	Motor 12 --
ISIG_MOTOR13_HOME	Motor 13 Home
ISIG_MOTOR13_PLUS	Motor 13 ++
ISIG_MOTOR13_MINUS	Motor 13 --
ISIG_MOTOR14_HOME	Motor 14 Home

ISIG_MOTOR14_PLUS	Motor 14 ++
ISIG_MOTOR14_MINUS	Motor 14 --
ISIG_MOTOR15_HOME	Motor 15 Home
ISIG_MOTOR15_PLUS	Motor 15 ++
ISIG_MOTOR15_MINUS	Motor 15 --
ISIG_MOTOR16_HOME	Motor 16 Home
ISIG_MOTOR16_PLUS	Motor 16 ++
ISIG_MOTOR16_MINUS	Motor 16 --
ISIG_MOTOR17_HOME	Motor 17 Home
ISIG_MOTOR17_PLUS	Motor 17 ++
ISIG_MOTOR17_MINUS	Motor 17 --
ISIG_MOTOR18_HOME	Motor 18 Home
ISIG_MOTOR18_PLUS	Motor 18 ++
ISIG_MOTOR18_MINUS	Motor 18 --
ISIG_MOTOR19_HOME	Motor 19 Home
ISIG_MOTOR19_PLUS	Motor 19 ++
ISIG_MOTOR19_MINUS	Motor 19 --
ISIG_MOTOR20_HOME	Motor 20 Home
ISIG_MOTOR20_PLUS	Motor 20 ++
ISIG_MOTOR20_MINUS	Motor 20 --
ISIG_MOTOR21_HOME	Motor 21 Home
ISIG_MOTOR21_PLUS	Motor 21 ++
ISIG_MOTOR21_MINUS	Motor 21 --
ISIG_MOTOR22_HOME	Motor 22 Home
ISIG_MOTOR22_PLUS	Motor 22 ++
ISIG_MOTOR22_MINUS	Motor 22 --
ISIG_MOTOR23_HOME	Motor 23 Home
ISIG_MOTOR23_PLUS	Motor 23 ++
ISIG_MOTOR23_MINUS	Motor 23 --
ISIG_MOTOR24_HOME	Motor 24 Home
ISIG_MOTOR24_PLUS	Motor 24 ++
ISIG_MOTOR24_MINUS	Motor 24 --
ISIG_MOTOR25_HOME	Motor 25 Home
ISIG_MOTOR25_PLUS	Motor 25 ++
ISIG_MOTOR25_MINUS	Motor 25 --

ISIG_MOTOR26_HOME	Motor 26 Home
ISIG_MOTOR26_PLUS	Motor 26 ++
ISIG_MOTOR26_MINUS	Motor 26 --
ISIG_MOTOR27_HOME	Motor 27 Home
ISIG_MOTOR27_PLUS	Motor 27 ++
ISIG_MOTOR27_MINUS	Motor 27 --
ISIG_MOTOR28_HOME	Motor 28 Home
ISIG_MOTOR28_PLUS	Motor 28 ++
ISIG_MOTOR28_MINUS	Motor 28 --
ISIG_MOTOR29_HOME	Motor 29 Home
ISIG_MOTOR29_PLUS	Motor 29 ++
ISIG_MOTOR29_MINUS	Motor 29 --
ISIG_MOTOR30_HOME	Motor 30 Home
ISIG_MOTOR30_PLUS	Motor 30 ++
ISIG_MOTOR30_MINUS	Motor 30 --
ISIG_MOTOR31_HOME	Motor 31 Home
ISIG_MOTOR31_PLUS	Motor 31 ++
ISIG_MOTOR31_MINUS	Motor 31 --

Output Signals

Signal	Description
OSIG_DIGTRIGGER	Digitize Trigger
OSIG_SPINDLEON	Spindle On
OSIG_SPINDLEFWD	Spindle Fwd
OSIG_SPINDLEREV	Spindle Rev
OSIG_COOLANTON	Coolant On
OSIG_MISTON	Mist On
OSIG_CHARGE	Charge Pump #1
OSIG_CHARGE2	Charge Pump #2
OSIG_CURRENTHILOW	Current Hi/Low
OSIG_ENABLE0	Enable #0
OSIG_ENABLE1	Enable #1
OSIG_ENABLE2	Enable #2
OSIG_ENABLE3	Enable #3
OSIG_ENABLE4	Enable #4
OSIG_ENABLE5	Enable #5
OSIG_ENABLE6	Enable #6
OSIG_ENABLE7	Enable #7
OSIG_ENABLE8	Enable #8
OSIG_ENABLE9	Enable #9
OSIG_ENABLE10	Enable #10
OSIG_ENABLE11	Enable #11
OSIG_ENABLE12	Enable #12
OSIG_ENABLE13	Enable #13
OSIG_ENABLE14	Enable #14
OSIG_ENABLE15	Enable #15
OSIG_ENABLE16	Enable #16
OSIG_ENABLE17	Enable #17
OSIG_ENABLE18	Enable #18
OSIG_ENABLE19	Enable #19

OSIG_ENABLE20	Enable #20
OSIG_ENABLE21	Enable #21
OSIG_ENABLE22	Enable #22
OSIG_ENABLE23	Enable #23
OSIG_ENABLE24	Enable #24
OSIG_ENABLE25	Enable #25
OSIG_ENABLE26	Enable #26
OSIG_ENABLE27	Enable #27
OSIG_ENABLE28	Enable #28
OSIG_ENABLE29	Enable #29
OSIG_ENABLE30	Enable #30
OSIG_ENABLE31	Enable #31
OSIG_XLIMITPLUS	X ++
OSIG_XLIMITMINUS	X --
OSIG_XHOME	X Home
OSIG_YLIMITPLUS	Y ++
OSIG_YLIMITMINUS	Y --
OSIG_YHOME	Y Home
OSIG_ZLIMITPLUS	Z ++
OSIG_ZLIMITMINUS	Z --
OSIG_ZHOME	Z Home
OSIG_ALIMITPLUS	A ++
OSIG_ALIMITMINUS	A --
OSIG_AHOME	A Home
OSIG_BLIMITPLUS	B ++
OSIG_BLIMITMINUS	B --
OSIG_BHOME	B Home
OSIG_CLIMITPLUS	C ++
OSIG_CLIMITMINUS	C --
OSIG_CHOME	C Home
OSIG_OUTPUT0	Output #0
OSIG_OUTPUT1	Output #1
OSIG_OUTPUT2	Output #2
OSIG_OUTPUT3	Output #3
OSIG_OUTPUT4	Output #4

OSIG_OUTPUT5	Output #5
OSIG_OUTPUT6	Output #6
OSIG_OUTPUT7	Output #7
OSIG_OUTPUT8	Output #8
OSIG_OUTPUT9	Output #9
OSIG_OUTPUT10	Output #10
OSIG_OUTPUT11	Output #11
OSIG_OUTPUT12	Output #12
OSIG_OUTPUT13	Output #13
OSIG_OUTPUT14	Output #14
OSIG_OUTPUT15	Output #15
OSIG_OUTPUT16	Output #16
OSIG_OUTPUT17	Output #17
OSIG_OUTPUT18	Output #18
OSIG_OUTPUT19	Output #19
OSIG_OUTPUT20	Output #20
OSIG_OUTPUT21	Output #21
OSIG_OUTPUT22	Output #22
OSIG_OUTPUT23	Output #23
OSIG_OUTPUT24	Output #24
OSIG_OUTPUT25	Output #25
OSIG_OUTPUT26	Output #26
OSIG_OUTPUT27	Output #27
OSIG_OUTPUT28	Output #28
OSIG_OUTPUT29	Output #29
OSIG_OUTPUT30	Output #30
OSIG_OUTPUT31	Output #31
OSIG_OUTPUT32	Output #32
OSIG_OUTPUT33	Output #33
OSIG_OUTPUT34	Output #34
OSIG_OUTPUT35	Output #35
OSIG_OUTPUT36	Output #36
OSIG_OUTPUT37	Output #37
OSIG_OUTPUT38	Output #38

OSIG_OUTPUT39	Output #39
OSIG_OUTPUT40	Output #40
OSIG_OUTPUT41	Output #41
OSIG_OUTPUT42	Output #42
OSIG_OUTPUT43	Output #43
OSIG_OUTPUT44	Output #44
OSIG_OUTPUT45	Output #45
OSIG_OUTPUT46	Output #46
OSIG_OUTPUT47	Output #47
OSIG_OUTPUT48	Output #48
OSIG_OUTPUT49	Output #49
OSIG_OUTPUT50	Output #50
OSIG_OUTPUT51	Output #51
OSIG_OUTPUT52	Output #52
OSIG_OUTPUT53	Output #53
OSIG_OUTPUT54	Output #54
OSIG_OUTPUT55	Output #55
OSIG_OUTPUT56	Output #56
OSIG_OUTPUT57	Output #57
OSIG_OUTPUT58	Output #58
OSIG_OUTPUT59	Output #59
OSIG_OUTPUT60	Output #60
OSIG_OUTPUT61	Output #61
OSIG_OUTPUT62	Output #62
OSIG_OUTPUT63	Output #63
OSIG_RUNNING_GCODE	Gcode Running
OSIG_FEEDHOLD	Feed Hold
OSIG_BLOCK_DELETE	Block Delete
OSIG_SINGLE_BLOCK	Single Block
OSIG_REVERSE_RUN	Reverse Run
OSIG_OPT_STOP	Opt Stop
OSIG_MACHINE_ENABLED	Machine Enabled
OSIG_TOOL_CHANGE	Tool Change
OSIG_DIST_TOGO	Dist To Go

OSIG_MACHINE_CORD	Machine Coord
OSIG_SOFTLIMITS_ON	Softlimits On
OSIG_JOG_INC	Jog Inc
OSIG_JOG_CONT	Jog Cont
OSIG_JOG_ENABLED	Jog Enabled
OSIG_JOG_MPG	Jog MPG
OSIG_HOMED_X	X Homed
OSIG_HOMED_Y	Y Homed
OSIG_HOMED_Z	Z Homed
OSIG_HOMED_A	A Homed
OSIG_HOMED_B	B Homed
OSIG_HOMED_C	C Homed
OSIG_DWELL	Dwell
OSIG_TP_MOUSE_DN	Toolpath Mouse Down
OSIG_LIMITOVER	Limit Override
OSIG_ALARM	Alarm
OSIG_PRTSF	Parts Finished

mcSignalEnable

C/C++ Syntax:

```
int mcSignalEnable(
 HMCSIG hSig,
 BOOL enabled);
```

LUA Syntax:

```
rc = mc.mcSignalEnable(
 number hSig,
 number enabled)
```

Description: Enable or disable a signal mapping.

Parameters:

Parameter	Description
hSig	A sginal handle obtained from mcSignalGetHandle().
enabled	TRUE to enable, FALSE to disable.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The signal parameter was 0.

Notes:
The hSig parameter **MUST** be a valid signal handle. Otherwise, the function will crash.
Signals have to be mapped to an IO handle before they can be enabled. It is important to note that the signal itself is always enabled. What is enabled it the signal mapping.

Usage:

```
simControl::simControl(MINSTANCE mInst, HMCPLUG id)
{
 m_cid = mInst;
 m_id = id;
 m_timer = new simTimer(this);
 m_cycletime = .001;
 HMCSIG hSig;

 mcDeviceRegister(m_cid, m_id, "Sim0", "Simulation Device", DEV_TYPE_MOTION | DE
```

```
 mcIoRegister(m_hDev, "Input0", "Input0", IO_TYPE_INPUT, &m;_Input0);
 mcIoRegister(m_hDev, "Output0", "Output0", IO_TYPE_OUTPUT, &m;_Output0)

 if (mcSignalGetHandle(m_cid, ISIG_INPUT1, int sigid, &hSig;) == MERROR_NOERROR)
  if (mcSignalMap(hSig, m_Input0) == MERROR_NOERROR) {
   // Signal mapped successfuly
mcSignalEnable(hSig, TRUE); // Enable the signal.
  }
 }
}
```

mcSignalGetHandle

C/C++ Syntax:

```
mcSignalGetHandle(
 MINSTANCE mInst,
 int sigid,
 HMCSIG *hSig);
```

LUA Syntax:

```
hSig, rc = mc.mcSignalGetHandle(
 number mInst,
 number sigid)
```

Description: Register the IO device in Mach4Core so it can be mapped by the user and other plugins

Parameters:

Parameter	Description
mInst	The controller instance.
sigid	The integer ID of the desired signal.
hSig	A pointer to a HMCSIG to receive the sginal handle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_SIGNAL_NOT_FOUND	The signal identified by sigid was not found.

Notes:
None.

Usage:

```
simControl::simControl(MINSTANCE mInst, HMCPLUG id)
{
 m_cid = mInst;
 m_id = id;
 m_timer = new simTimer(this);
```

```
 m_cycletime = .001;
 HMCSIG hSig;

 mcDeviceRegister(m_cid, m_id, "Sim0", "Simulation Device", DEV_TYPE_MOTION | DE

 mcIoRegister(m_hDev, "Input0", "Input0", IO_TYPE_INPUT, &m;_Input0);
 mcIoRegister(m_hDev, "Output0", "Output0", IO_TYPE_OUTPUT, &m;_Output0)

 if (mcSignalGetHandle(m_cid, ISIG_INPUT1, &hSig;) == MERROR_NOERROR) {
  if (mcSignalMap(hSig, m_Input0) == MERROR_NOERROR) {
   // Signal mapped successfuly.
   mcSignalEnable(hSig, true); // Enable the signal.
  }
 }
}
```

mcSignalGetInfo

C/C++ Syntax:

```
mcSignalGetInfo(
 HMCSIG hSig,
 int *enabled,
 char *name,
 size_t namelen,
 char *desc,
 size_t desclen,
 int *activelow);
```

LUA Syntax:

```
enabled, name, desc, activelow, rc = mc.mcSignalGetInfo(number hSig)
```

Description: Return infromation about a signal.

Parameters:

Parameter	Description
hSig	The handle for the signal.
enabled	The address of an integer to receive the enabled status of the signal.
name	A buffer to receive the name of the signal.
namelen	A value specifiying the length of the name buffer.
desc	A buffer to receive the description of the signal.
desclen	A value specifiying the length of the description buffer.
activelow	The address of an integer to receive the active low status of the signal.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The hSig parameter was 0.

Notes:
The hSig parameter **MUST** be a valid signal handle. Otherwise, the function will crash.

Usage:

```
HMCSIG hSig = 0;
int enabled = 0;
int nameLen = 20
char nameBuf[nameLen];
int descLen = 40;
char descBuf[descLen];
int activeLow = 0;

while (mcSignalGetNextHandle(0, SIG_TYPE_INPUT, hSig, &hSig;) == MERROR_NOERROR)
 if (hSig != 0) {
  mcSignalGetInfo(hSig, &enabled;, nameBuf, nameLen, descBuf, descLen, &activeLo
 } else {
  break;
 }
}
```

mcSignalGetInfoStruct

C/C++ Syntax:

```
mcSignalGetInfoStruct(
 HMCSIG hSig,
 siginfo_t *siginf);
```

LUA Syntax:

```
N/A
```

Description: Return infromation about a signal.

Parameters:

Parameter	Description
hSig	The handle for the signal.
siginf	The adress of a siginfo struct to receive the signal information.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
No Error.	
MERROR_INVALID_ARG	The hSig parameter was 0.

Notes:
The hSig parameter **MUST** be a valid signal handle. Otherwise, the function will crash.

```
struct siginfo {
 char sigName[80];
 char sigDesc[80];
 int  sigEnabled;
 int  sigActiveLow;
 HMCIO sigMappedIoHandle;
};
typedef struct siginfo siginfo_t;
```

Usage:

```
HMCSIG hSig = 0;
```

```
siginfo_t siginf;

while (mcSignalGetNextHandle(0, SIG_TYPE_INPUT, hSig, &hSig;) == MERROR_NOERROR)
 if (hSig != 0) {
  mcSignalGetInfoStruct(hSig, &siginf;);
 } else {
  break;
 }
}
```

mcSignalGetNextHandle

C/C++ Syntax:

```
mcSignalGetNextHandle(
 MINSTANCE mInst,
 int sigtype,
 HMCSIG startSig,
 HMCSIG *hSig);
```

LUA Syntax:

```
hSig, rc = mc.mcSignalGetNextHandle(
 number mInst,
 number sigtype,
 number startSig)
```

Description: Provides a means of looping through the available signals.

Parameters:

Parameter	Description
mInst	The controller instance.
sigtype	The signal type for which to look (SIG_TYPE_INPUT or SIG_TYPE_OUTPUT).
startSig	The current signal handle.
hSig	A pointer to a HMCSIG to receive the signal handle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	sigtype was not SIG_TYPE_INPUT or SIG_TYPE_OUTPUT.
MERROR_NODATA	No more signals can be found.

Notes:
To start, call mcSignalGetNextHandle() with startSig = 0.

Usage:

```
HMCSIG hSig = 0;

while (mcSignalGetNextHandle(0, SIG_TYPE_INPUT, hSig, &hSig;) == MERROR_NOERROR)
    if (hSig != 0) {
        // Do something with hSig.
    } else {
        break;
    }
}
```

mcSignalGetState

C/C++ Syntax:

```
mcSignalGetState(
 HMCSIG hSig,
 BOOL *state)
```

LUA Syntax:

```
state, rc = mc.mcSignalGetState(
 number hSig)
```

Description: Set the state of a signal.

Parameters:

Parameter	Description
hSig	The handle for the signal.
state	Defines the desired state of the signal. TRUE is active

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The hSig parameter was 0 or invalid.

Notes:
The hSig parameter **MUST** be a valid signal handle. Otherwise, the function will crash.

Usage:

```
HMCSIG hSig = 0;
BOOL sigState = FALSE;

// Get the state of all input signals.
while (mcSignalGetNextHandle(0, SIG_TYPE_INPUT, hSig, &hSig;) == MERROR_NOERROR)
    if (hSig != 0) {
        mcSignalGetState(hSig, &sigState;);
    } else {
        break;
    }
}
```

mcSignalMap

C/C++ Syntax:

```
mcSignalMap(
 HMCSIG hSig,
 HMCIO hIo);
```

LUA Syntax:

```
rc = mc.mcSignalMap(
 number hSig,
 number hIo)
```

Description: Map a core signal to a registered I/O handle.

Parameters:

Parameter	Description
hSig	A sginal handle obtained from mcSignalGetHandle().
hIo	A registered IO handle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The signal parameter was 0.

Notes:
The hSig and hIo parameters **MUST** be a valid handles. Otherwise, the function will crash.
Mapping an input IO handle to an output signal handle or an output IO handle to an input signal handle is a programming error.

Usage:

```
simControl::simControl(MINSTANCE mInst, HMCPLUG id)
{
 m_cid = mInst;
 m_id = id;
 m_timer = new simTimer(this);
 m_cycletime = .001;
 HMCSIG hSig;

 mcDeviceRegister(m_cid, m_id, "Sim0", "Simulation Device", DEV_TYPE_MOTION | DE
```

```
 mcIoRegister(m_hDev, "Input0", "Input0", IO_TYPE_INPUT, &m;_Input0);
 mcIoRegister(m_hDev, "Output0", "Output0", IO_TYPE_OUTPUT, &m;_Output0)

 if (mcSignalGetHandle(m_cid, ISIG_INPUT1, &hSig;) == MERROR_NOERROR) {
  if (mcSignalMap(hSig, m_Input0) == MERROR_NOERROR) {
   // Signal mapped successfuly
  }
 }
}
```

mcSignalSetActiveLow

C/C++ Syntax:

```
mcSignalSetActiveLow(
 HMCSIG hSig,
 BOOL activelow);
```

LUA Syntax:

```
rc = mc.mcSignalSetActiveLow(
 number hSig,
 number activelow)
```

Description: Set the state of a signal.

Parameters:

Parameter	Description
hSig	The handle for the signal.
activelow	TRUE to set the signal as negated.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The hSig parameter was 0 or invalid.

Notes:
The hSig parameter **MUST** be a valid signal handle. Otherwise, the function will crash.

Usage:

```
HMCSIG hSig = 0;
BOOL activeLow = TRUE;

// Set all input signals active low.
while (mcSignalGetNextHandle(0, SIG_TYPE_INPUT, hSig, &hSig;) == MERROR_NOERROR)
 if (hSig != 0) {
  mcSignalSetActiveLow(hSig, activeLow);
 } else {
  break;
 }
}
```

mcSignalSetState

C/C++ Syntax:

```
mcSignalSetState(
 HMCSIG hSig,
 BOOL enabled);
```

LUA Syntax:

```
rc = mc.mcSignalSetState(
 number hSig,
 number enabled)
```

mcSignalSetState(HMCSIG hSig, bool enabled); **Description:**
Set the state of a signal.

Parameters:

Parameter	Description
hSig	The handle for the signal.
enabled	TRUE is active

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The hSig parameter was 0.

Notes:
The hSig parameter **MUST** be a valid signal handle. Otherwise, the function will crash.

Usage:

```
HMCSIG hSig = 0;

// Set all output signals inactive.
while (mcSignalGetNextHandle(0, SIG_TYPE_OUTPUT, hSig, &hSig;) == MERROR_NOERROR)
 if (hSig != 0) {
  mcSignalSetState(hSig, FALSE);
 } else {
  break;
 }
}
```

mcSignalUnmap

C/C++ Syntax:

```
mcSignalUnmap(
 HMCSIG hSig);
```

LUA Syntax:

```
rc = mc.mcSignalUnmap(
 number hSig)
```

Description: Unmap any I/O objects from the specified core signal.

Parameters:

Parameter	Description
hSig	A sginal handle obtained from mcSignalGetHandle().

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The signal parameter was 0 or invalid.

Notes:
The hSig parameters **MUST** be a valid handle. Otherwise, the function will crash.

Usage:

```
MINSTANCE mInst = 0;
HMCSIG hSig;

if (mcSignalGetHandle(mInst, ISIG_INPUT1, &hSig;) == MERROR_NOERROR) {
 if (mcSignalUnmap(hSig) == MERROR_NOERROR) {
  // Signal now has no I/O mappings
 }
}
```

mcSignalWait

C/C++ Syntax:

```
mcSignalWait(
 MINSTANCE mInst,
 int sigId,
 int waitMode,
 double timeoutSecs);
```

LUA Syntax:

```
rc = mc.mcSignalWait(
 number mInst,
 number sigId,
 number waitMode,
 number timeoutSecs);
```

Description: Wait on a signal to change state.

Parameters:

Parameter	Description
hSig	A sginal handle obtained from mcSignalGetHandle().
sigId	A valid signal ID.
waitMode	An integer specifying whether to wait on the signal to go high (WAIT_MODE_HIGH) or low (WAIT_MODE_LOW).
timeoutSecs	A double specifying a timeout period.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**sigId** or **wiatMode** is out of range or **timeoutSecs** is negative.
MERROR_NOT_ENABLED	The control is not enabled.
MERROR_TIMED_OUT	The timeout period was reached without a change of state.

Notes:

Specifying 0 for **timeoutSecs** will wait indefinitely until the signal changes state. Decimals are allowed. e.g. .5 is half of a second.)

If the control is diabled while waiting on a signal to change state, the function stops waiting and returns MERROR_NOT_ENABLED.

Usage:

```
MINSTANCE mInst = 0;
HMCSIG hSig;
int rc;

rc = mcSignalWait(mInst, ISIG_INPUT1, WAIT_MODE_HIGH, .5);
switch (rc) {
case MERROR_NOERROR:
 // Signal changed state from low to high
 break;
case MERROR_TIMED_OUT:
 // The signal didn't change state in the alotted time.
 break;
case MERROR_NOT_ENABLED:
 // The control was not enabled at the time of the
   function call or the control was disabled during the function call.
 break;
}
```

Mach Registers

Mach registers are multi-purpose storage locations that can be defined (registered) by plugins. The parent to a register is a plugin device. Registers can contain data of any type, even string data.

Registers provide a uniquie storage location that is owned by the plugin that creates it. This is in contrast to Mach 3 DROs and LEDs that could be overwritten if two plugins wrote to the same DRO number not knowing that the other plugin also used that same DRO number. However, registers can be written and read from any script or plugin. But the name of the register has to be known!

The maximum string length is 4096 bytes for registers other than **REG_TYPE_COMMAND** type registers and 1024 bytes for the **REG_TYPE_COMMAND** type registers.

When a register is registered to the system, you must specify the register type. The different types are discussed below:

REG_TYPE_NONE specifies a register that is not typed. This kind of register is basically just a data container that will not notify any process if the data it stores changes.

REG_TYPE_INPUT specifies a register that can be mapped to a Mach input signal. Mach and any GUI front end is notified if the data in this type of register is changed with a **MSG_REG_CHANGED** message. This type of register usually contains boolean data.

REG_TYPE_OUTPUT specifies a register that can be mapped to a Mach output signal. The plugin that owns this register is notified with a **MSG_REG_CHANGED** synchronous message. All other plugins and the GUI are notified asynchronously with **MSG_REG_CHANGED**.

REG_TYPE_HOLDING specifies a general purpose register that cannot be mapped to either an input or output signal. The plugin that owns this register is notified with a **MSG_REG_CHANGED** synchronous message. All other plugins and the GUI are notified asynchronously with **MSG_REG_CHANGED**. This type of register usually contains boolean data.

REG_TYPE_COMMAND specifies a special purpose register that can be used to implement a simple command/response communication mechanism. Only the plugin that owns the register is notified with a **MSG_REG_COMMAND** synchronous message. It works with

- mcRegSendCommand

on the client side of the communication and

- mcRegGetCommand

and

- mcRegSetResponse

on the plguin side.

REG_TYPE_ENCODER specifies a register that can be mapped to a MPG or used to display the encoder counts in the GUI. Only Mach and the GUI are notified with a **MSG_REG_CHANGED** synchronous message.

- mcRegGetCommand
- mcRegGetHandle
- mcRegGetInfo
- mcRegGetInfoStruct
- mcRegGetNextHandle
- mcRegGetUserData
- mcRegGetValue
- mcRegGetValueLong
- mcRegGetValueString
- mcRegGetValueStringClear
- mcRegRegister
- mcRegSendCommand
- mcRegSetDesc
- mcRegSetName
- mcRegSetResponse
- mcRegSetType
- mcRegSetUserData
- mcRegSetValue
- mcRegSetValueLong
- mcRegSetValueString
- mcRegUnregister

mcRegGetCommand

C/C++ Syntax:

```
int mcRegGetCommand(
  HMCREG hReg,
  char *cmd,
  size_t cmdLen)
```

LUA Syntax:

```
N/A
```

Description: Retrieve a command from the given register.

Parameters:

Parameter	Description
hReg	A HMCREG specifying the register handle.
cmd	A string buffer to receive the command.
cmdLen	The sized of the command buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_REG_NOT_FOUND	The **hReg** parameter was 0 or invlaid.
MERROR_INVALID_ARG	**cmd** is NULL.
MERROR_INVALID_TYPE	The register is not of type **REG_TYPE_COMMAND**.

Notes:
The function will not return a command larger than 1024 bytes.

See Also:

- mcRegSendCommand

and

- mcRegSetResponse

Usage:

```
// Retrieve the register command.
int simControl::ProcessMsg(long msg, long param1, long param2)
{
 HMCREG hReg = (HMCREG)param1;
 long RegVal;
 switch (msg) {
 case MSG_REG_COMMAND:
  mcRegGetValueLong(hReg, &RegVal;);
  if (hReg = m_RegCommand) { // Is this our command register?
   char command[1024];
   mcRegGetCommand(hReg, command, sizeof(command));
   wxString cmd(command);
   cmd.MakeUpper();
   if (cmd == wxT("THC ON")) {
    m_thc = true;
    mcRegSetResponse(hReg, "OK");
   } else if (cmd == wxT("THC OFF")) {
    m_thc = false;
    mcMotionSync(m_cid);
    mcRegSetResponse(hReg, "OK");
   } else if (cmd == wxT("THC STATUS")) {
    if (m_thc) {
     mcRegSetResponse(hReg, "1");
    } else {
     mcRegSetResponse(hReg, "0");
    }
   }
  }
  break;
 default:
  ;
 }
 return(MERROR_NOERROR);
}
```

mcRegGetHandle

C/C++ Syntax:

```
int mcRegGetHandle(
  MINSTANCE mInst,
  const char *path,
  HMCREG *hReg);
```

LUA Syntax:

```
hReg, rc = mc.mcRegGetHandle(
  number mInst,
  string path)
```

Description: Retrieve a handle to a register give its path.

Parameters:

Parameter	Description
mInst	The controller instance.
path	A string buffer specifying the register path.
hReg	The address of a HMCREG to receive the register handle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**path** or **hReg** is NULL.

Notes:
None.

Usage:

```
// Get the register handle.
MINSTANCE mInst = 0;
HMCREG hReg;
int rc;
double value;
if (mcRegGetHandle(mInst, "core/global/Version", &hReg;) == MERROR_NOERROR) {
```

```
 rc = mcRegGetValue(hReg, &value;);
}
```

mcRegGetInfo

C/C++ Syntax:

```
int mcRegGetInfo(
  HMCREG hReg,
  char *nameBuf,
  size_t nameBuflen,
  char *descBuf,
  size_t descBuflen,
  int *type,
  HMCDEV *hDev);
```

LUA Syntax:

```
nameBuf, descBuf, type, hDev, rc = mc.mcRegGetInfo(
  number hReg)
```

Description: Return infromation about a register.

Parameters:

Parameter	Description
hReg	A HMCREG specifying the register handle.
nameBuf	A string buffer to receive the name of the register.
nameBuflen	The length of the name buffer.
descBuf	A string buffer to receive the description of the register.
descBuflen	The length of the description buffer.
type	The address of an integer to receive the register type.
hDev	The address of an integer to receive the register handle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_REG_NOT_FOUND	The **hReg** parameter was 0 or invlaid.

Notes:
The hReg parameter **MUST** be a valid signal handle. Otherwise, the function will crash. **nameBuf** ,**descBuf**, **type**, or **hDev** can be NULL depending on the amout of information required.

Usage:

```
HMCREG hReg = 0;
char name[80];
char desc[80];
int type;
HMCDEV hDev;

while (mcSignalGetNextHandle(hDev, hReg, &hReg;) == MERROR_NOERROR) {
 if (hSig != 0) {
  // Get the info on the register.
  mcRegGetInfo(hReg, name, sizeof(name), desc, sizeof(desc), &type;, &hDev;));
 } else {
  break;
 }
}
```

mcRegGetInfoStruct

C/C++ Syntax:

```
mcRegGetInfoStruct(
  HMCREG hReg,
  reginfo_t *reginf);
```

LUA Syntax:

```
N/A
```

Description: Return infromation about a register.

Parameters:

Parameter	Description
hReg	A HMCREG specifying the register handle.
reginf	The adress of a reginfo struct to receive the register information.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_REG_NOT_FOUND	The **hReg** parameter was 0 or invlaid.
MERROR_INVALID_ARG	**reginf** is NULL

Notes:
The hReg parameter **MUST** be a valid signal handle. Otherwise, the function will crash.

```
struct reginfo {
 char regName[80];
 char regDesc[80];
 int  regType;
 HMCDEV regDev;
 void *regUserData;
 int regInput;
};
typedef struct reginfo reginfo_t;
```

Usage:

```
HMCREG hReg = 0;
reginfo_t reginf;

while (mcSignalGetNextHandle(hDev, hReg, &hReg;) == MERROR_NOERROR) {
 if (hSig != 0) {
  // Get the info on the register.
  mcRegGetInfoStruct(hReg, ®inf;);
 } else {
  break;
 }
}
```

mcRegGetNextHandle

C/C++ Syntax:

```
int mcRegGetNextHandle(
  HMCDEV hDev,
  HMCREG startReg,
  HMCREG *hReg);
```

LUA Syntax:

```
hReg, rc = mc.mcRegGetNextHandle(
  number hDev,
  number startReg)
```

Description: Provides a means of looping through the available registers.

Parameters:

Parameter	Description
hDev	the register's parent device handle.
startReg	The starting register handle.
hReg	The address of a hMCREG to receive the next register handle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_DEVICE_NOT_FOUND	The **hDev** parameter was 0 or invlaid.
MERROR_INVALID_ARG	**hReg** is NULL.
MERROR_NODATA	No more register handles can be found.

Notes:
To start, call mcRegGetNextHandle() with startReg = 0. hDev MUST be valid or the application can crash.

Usage:

```
HMCSIG hReg = 0;
HMCDEV hDev;
mcDeviceGetHandle(0, 0,&hDev;);
while (mcRegisterGetNextHandle(hDev, hReg, &hReg;) == MERROR_NOERROR) {
 if (hSig != 0) {
  // do something with hReg.
 } else {
  break;
 }
}
```

mcRegGetUserData

C/C++ Syntax:

```
int mcRegGetUserData(
  HMCREG hReg,
  void **data);
```

LUA Syntax:

```
N/A
```

Description: Retrieve the user data pointer for the hReg.

Parameters:

Parameter	Description
hReg	A HMCREG specifying the register handle.
data	The address of a void pointer receive the user data pointer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_REG_NOT_FOUND	The **hReg** parameter was 0 or invlaid.
MERROR_INVALID_ARG	**data** is NULL.

Notes:
The hReg parameter **MUST** be a valid register handle. Otherwise, the function will crash.
This function allows the programmer to retreive a pointer to asscoiated private data that is local to the register's parent plugin/device.

Usage:

```
struct myDataStruct {
 int myInt;
 char myString[80];
 ...
};

HMCREG hReg;
int mInst=0;
```

```
myDataStruct *data = NULL;
void *dataPtr;
// Get the handle to holding register 1 on SIM0.
if (mcRegGetHandle(mInst, "SIM0/HoldingReg1", &hReg;) == MERROR_NOERROR) {
 mcRegGetUserData(hReg, &dataPtr;);
 data = (myDataStruct *) dataPtr;
}
```

mcRegGetValue

C/C++ Syntax:

```
int mcRegGetValue(
  HMCREG hReg,
  double *value);
```

LUA Syntax:

```
value, rc = mcRegGetValue(
  number hReg)
```

Description: Retrieve a register's double value.

Parameters:

Parameter	Description
hReg	A HMCREG specifying the register handle.
value	The address of a double to receive the value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_REG_NOT_FOUND	The **hReg** parameter was 0 or invlaid.
MERROR_INVALID_ARG	**value** is NULL.

Notes:
None.

Usage:

```
// Get the register value.
MINSTANCE mInst = 0;
HMCREG hReg;
int rc;
double value;
if (mcRegGetHandle(mInst, "core/global/Version", &hReg;) == MERROR_NOERROR) {
 rc = mcRegGetValue(hReg, &value;);
}
```

mcRegGetValueLong

C/C++ Syntax:

```
int mcRegGetValueLong(
  HMCREG hReg,
  double *value);
```

LUA Syntax:

```
value, rc = mcRegGetValueLong(
  number hReg)
```

Description: Retrieve a register's long value.

Parameters:

Parameter	Description
hReg	A HMCREG specifying the register handle.
value	The address of a long to receive the value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_REG_NOT_FOUND	The **hReg** parameter was 0 or invlaid.
MERROR_INVALID_ARG	**value** is NULL.

Notes:
None.

Usage:

```
// Get the register value.
MINSTANCE mInst = 0;
HMCREG hReg;
int rc;
long value;
if (mcRegGetHandle(mInst, "core/global/Version", &hReg;) == MERROR_NOERROR) {
 rc = mcRegGetValueLong(hReg, &value;);
}
```

mcRegGetValueString

C/C++ Syntax:

```
int mcRegGetValueString(
  HMCREG hReg,
  char *buf,
  size_t bufSize);
```

LUA Syntax:

```
buf, rc = mcRegGetValueString(
  number hReg)
```

Description: Retrieve a register's string value.

Parameters:

Parameter	Description
hReg	A HMCREG specifying the register handle.
buf	A string buffer to receive the value.
bufSize	The length of the string buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_REG_NOT_FOUND	The **hReg** parameter was 0 or invlaid.
MERROR_INVALID_ARG	**buf** is NULL.

Notes:
None.

Usage:

```
// Get the register value.
MINSTANCE mInst = 0;
HMCREG hReg;
int rc;
char value[80];
if (mcRegGetHandle(mInst, "core/global/Version", &hReg;) == MERROR_NOERROR) {
 rc = mcRegGetValueString(hReg, &value;, sizeof(value));
}
```

mcRegGetValueStringClear

C/C++ Syntax:

```
int mcRegGetValueStringClear(
  HMCREG hReg,
  char *buf,
  size_t bufSize);
```

LUA Syntax:

```
buf, rc = mcRegGetValueString(
  number hReg)
```

Description: Retrieve a register's string value and then clear the register.

Parameters:

Parameter	Description
hReg	A HMCREG specifying the register handle.
buf	A string buffer to receive the value.
bufSize	The length of the string buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_REG_NOT_FOUND	The **hReg** parameter was 0 or invlaid.
MERROR_INVALID_ARG	**buf** is NULL.

Notes:
None.

Usage:

```
// Get the register value and clear it.
MINSTANCE mInst = 0;
HMCREG hReg;
int rc;
char value[80];
if (mcRegGetHandle(mInst, "core/global/Version", &hReg;) == MERROR_NOERROR) {
 rc = mcRegGetValueStringClear(hReg, &value;, sizeof(value));
}
```

mcRegRegister

C/C++ Syntax:

```
int mcRegRegister(
  HMCDEV hDev,
  const char *regName,
  const char *regDesc,
  int regType,
  HMCREG *hReg);
```

LUA Syntax:

```
N/A
```

Description: Adds a register to the core.

Parameters:

Parameter	Description
hDev	The handle of the register's parent device.
regName	A string buffer specifying the name of the register.
regDesc	A string buffer specifying the description of the register.
regType	REG_TYPE_NONE, REG_TYPE_INPUT, REG_TYPE_OUTPUT, REG_TYPE_HOLDING, REG_TYPE_COMMAND, REG_TYPE_ENCODER.
hReg	The address of a HMCREG that receives the register handle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_DEVICE_NOT_FOUND	The **hDev** parameter was 0 or invlaid.
MERROR_INVALID_ARG	**regName**, **regDesc**, or **hReg** is NULL.

Notes:
The hDev parameter **MUST** be a valid device handle. Otherwise, the function will crash.

Registers can contain data of any type, even string data. The maximum string length is 4096 bytes for registers other than **REG_TYPE_COMMAND** type registers and 1024 bytes for the **REG_TYPE_COMMAND** type registers.

REG_TYPE_NONE specifies a register that is not typed. This kind of register is basically just a data container that will not notify any process if the data it stores changes.

REG_TYPE_INPUT specifies a register that can be mapped to a Mach input signal. Mach and any GUI front end is notified if the data in this type of register is changed with a **MSG_REG_CHANGED** message. This type of register usually contains boolean data.

REG_TYPE_OUTPUT specifies a register that can be mapped to a Mach output signal. The plugin that owns this register is notified with a **MSG_REG_CHANGED** synchronous message. All other plugins and the GUI are notified asynchronously with **MSG_REG_CHANGED**.

REG_TYPE_HOLDING specifies a general purpose register that cannot be mapped to either an input or output signal. The plugin that owns this register is notified with a **MSG_REG_CHANGED** synchronous message. All other plugins and the GUI are notified asynchronously with **MSG_REG_CHANGED**. This type of register usually contains boolean data.

REG_TYPE_COMMAND specifies a special purpose register that can be used to implement a simple command/response communication mechanism. Only the plugin that owns the register is notified with a **MSG_REG_COMMAND** synchronous message. It works with

- mcRegSendCommand

on the client side of the communication and

- mcRegGetCommand

and

- mcRegSetResponse

on the plguin side.

REG_TYPE_ENCODER specifies a register that can be mapped to a MPG or used to display the encoder counts in the GUI. Only Mach and the GUI are notified with a **MSG_REG_CHANGED** synchronous message.

Usage:

```
HMCREG m_hReg;
HMCDEV m_hDev; // Contains the device handle from the device registration.
int rc = mcRegRegister(m_hDev, "HoldingReg1", "HoldingReg1", REG_TYPE_HOLDING, &
```

mcRegSendCommand

C/C++ Syntax:

```
int mcRegSendCommand(
  HMCREG hReg,
  const char *command,
  char *response,
  size_t responseLen);
```

LUA Syntax:

```
response, rc = mc.mcRegSendCommand(
  hReg,
  command)
```

Description: Sends a text command to a plugin register.

Parameters:

Parameter	Description
hReg	A HMCREG specifying the register handle.
command	A string buffer specifying the command to be sent.
response	A string buffer that receives the response.
responseLen	The length of the response buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	**command** or **response** is NULL.
MERROR_INVALID_TYPE	The register is not of type **REG_TYPE_COMMAND**.
MERROR_REG_NOT_FOUND	The **hReg** parameter was 0 or invlaid.

Notes:
This function would primarily be used in scripts to control device features or provide a direct communication mechanism to the device.

The buffer pointed to by cmd cannot exceed 1024 bytes.

See Also:

- mcRegGetCommand

and

- mcRegSetResponse

Usage:

```
// Send a command via a register.
MINSTANCE mInst = 0;
if (mcRegGetHandle(mInst, "Sim0/SimCommand", &hReg;) == MERROR_NOERROR) {
 char response[1024];
 rc = mcRegSendCommand(hReg, "THC ON", response, sizeof(response));
 if (rc == MERROR_NOERROR) {
  // check response.
 }
}
```

mcRegSetDesc

C/C++ Syntax:

```
int mcRegSetDesc(
  HMCREG hReg,
  const char *desc);
```

LUA Syntax:

```
rc = mc.mcRegSetDesc(
  number hReg,
  string desc)
```

Description: Sets the description of the given ragister.

Parameters:

Parameter	Description
hReg	A HMCREG specifying the register handle.
desc	A string buffer specifying the register description.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_REG_NOT_FOUND	The **hReg** parameter was 0 or invlaid.
MERROR_INVALID_ARG	**desc** is NULL.

Notes:
None.

Usage:

```
// Set/change the description of the register.
MINSTANCE mInst = 0;
if (mcRegGetHandle(mInst, "core/global/Version", &hReg;) == MERROR_NOERROR) {
 rc = mcRegSetDesc(hReg, "Core Version");
}
```

mcRegSetName

C/C++ Syntax:

```
int mcRegSetName(
  HMCREG hReg,
  const char *name);
```

LUA Syntax:

```
rc = mc.mcRegSetName(
  number hReg,
  string name)
```

Description: Set the name of the given register.

Parameters:

Parameter	Description
hReg	A HMCREG specifying the register handle.
name	A string buffer specifying the name of the register.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_REG_NOT_FOUND	The **hReg** parameter was 0 or invlaid.
MERROR_INVALID_ARG	**name** is NULL.

Notes:
None.

Usage:

```
// Set/change the name of the register.
MINSTANCE mInst = 0;
if (mcRegGetHandle(mInst, "core/global/Version", &hReg;) == MERROR_NOERROR) {
 rc = mcRegSetName(hReg, "Version2");
 // The register path is now "core/global/Version2"
}
```

mcRegSetResponse

C/C++ Syntax:

```
int mcRegSetResponse(
  HMCREG hReg,
  char *response);
```

LUA Syntax:

```
N/A
```

Description: Set the reponse to the command register.

Parameters:

Parameter	Description
hReg	A HMCREG specifying the register handle.
response	A string buffer that specifies the response to the command.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_TYPE	The register is not of type **REG_TYPE_COMMAND**.
MERROR_REG_NOT_FOUND	The **hReg** parameter was 0 or invlaid.

Notes:

See Also:

- mcRegSendCommand

and

- mcRegGetCommand

Usage:

```
// Set the response to a register command.
int simControl::ProcessMsg(long msg, long param1, long param2)
{
```

```
 HMCREG hReg = (HMCREG)param1;
 long RegVal;
 switch (msg) {
 case MSG_REG_COMMAND:
  mcRegGetValueLong(hReg, &RegVal;);
  if (hReg = m_RegCommand) { // Is this our command register?
   char command[1024];
   mcRegGetCommand(hReg, command, sizeof(command));
   wxString cmd(command);
   cmd.MakeUpper();
   if (cmd == wxT("THC ON")) {
    m_thc = true;
    mcRegSetResponse(hReg, "OK");
   } else if (cmd == wxT("THC OFF")) {
    m_thc = false;
    mcMotionSync(m_cid);
    mcRegSetResponse(hReg, "OK");
   } else if (cmd == wxT("THC STATUS")) {
    if (m_thc) {
     mcRegSetResponse(hReg, "1");
    } else {
     mcRegSetResponse(hReg, "0");
    }
   }
  }
  break;
 default:
  ;
 }
 return(MERROR_NOERROR);
}
```

mcRegSetType

C/C++ Syntax:

```
int mcRegSetType(
  HMCREG hReg,
  int regType);
```

LUA Syntax:

```
rc = mc.mcRegSetType(
  number hReg,
  number regType)
```

Description: Set the type of the given register.

Parameters:

Parameter	Description
hReg	A HMCREG specifying the register handle.
regType	An integer specifying the register type. (REG_TYPE_NONE, IO_TYPE_REG_INPUT, REG_TYPE_OUTPUT, REG_TYPE_HOLDING, REG_TYPE_COMMAND, REG_TYPE_ENCODER)

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_REG_NOT_FOUND	The **hReg** parameter was 0 or invlaid.
MERROR_INVALID_ARG	**regType** is out of range.

Notes:
None.

Usage:

```
// Set the register type.
MINSTANCE mInst = 0;
if (mcRegGetHandle(mInst, "core/global/Version", &hReg;) == MERROR_NOERROR) {
 rc = mcRegSetType(hReg, REG_TYPE_HOLDING);
}
```

mcRegSetUserData

C/C++ Syntax:

```
mcRegSetUserData(
  HMCREG hReg,
  void *data);
```

Description: Set the user pointer for hReg.

Parameters:

Parameter	Description
hReg	The IO handle.
value	The address of a void pointer so user can have there own data.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
No Error.	
MERROR_INVALID_ARG	The hReg parameter was 0.

Notes:
The hReg parameter **MUST** be a valid IO handle. Otherwise, the function will crash.
This function can be used to save a pointer to your own data.

Usage:

```
MSG_REG_CHANGED

HMCREG hReg;
int mInst=0;
RegisterStruct RegStruct;
//User data is set to where the data struct is saved.
// Get the handle to holding register 1 on SIM0.
mcRegGetHandle(mInst, "SIM0/HoldingReg1", &hReg;);
mcRegSetUserData(hReg, (void *)&RegStruct;);
```

mcRegSetValue

C/C++ Syntax:

```
int mcRegSetValue(
  HMCREG hReg,
  double value);
```

LUA Syntax:

```
rc = mc.mcRegSetValue(
  number hReg,
  number value)
```

Description: Sets the double value of the given register.

Parameters:

Parameter	Description
hReg	A HMCREG specifying the register handle.
value	A double specifying the register value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_REG_NOT_FOUND	The **hReg** parameter was 0 or invlaid.

Notes:
The hReg parameter **MUST** be a valid register handle. Otherwise, the function will crash.
This function should only be used upon a value change. It is a programming error otherwise and will cause messages to SPAM the core and GUI. MSG_REG_CHANGED is sent to the GUI and the plugin owning the device that owns the register.

Usage:

```
// Set the register value.
MINSTANCE mInst = 0;
if (mcRegGetHandle(mInst, "core/global/Version", &hReg;) == MERROR_NOERROR) {
 rc = mcRegSetValue(hReg, 4.0);
}
```

mcRegSetValueLong

C/C++ Syntax:

```
int mcRegSetValueLong(
  HMCREG hReg,
  double value);
```

LUA Syntax:

```
rc = mc.mcRegSetValueLong(
  number hReg,
  number value)
```

Description: Sets the long value of the given register.

Parameters:

Parameter	Description
hReg	A HMCREG specifying the register handle.
value	A long specifying the register value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_REG_NOT_FOUND	The **hReg** parameter was 0 or invlaid.

Notes:
The hReg parameter **MUST** be a valid register handle. Otherwise, the function will crash.
This function should only be used upon a value change. It is a programming error otherwise and will cause messages to SPAM the core and GUI. MSG_REG_CHANGED is sent to the GUI and the plugin owning the device that owns the register.

Usage:

```
// Set the register value.
MINSTANCE mInst = 0;
if (mcRegGetHandle(mInst, "core/global/Version", &hReg;) == MERROR_NOERROR) {
 rc = mcRegSetValueLong(hReg, 4);
}
```

mcRegSetValueString

C/C++ Syntax:

```
int mcRegSetValueString(
  HMCREG hReg,
  const char *value);
```

LUA Syntax:

```
rc = mc.mcRegSetValueString(
  number hReg,
  string value)
```

Description: Sets the string value of the given register.

Parameters:

Parameter	Description
hReg	A HMCREG specifying the register handle.
value	A string buffer specifying the register value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_REG_NOT_FOUND	The **hReg** parameter was 0 or invlaid.
MERROR_INVALID_ARG	**value** is NULL.

Notes:
The hReg parameter **MUST** be a valid register handle. Otherwise, the function will crash.
This function should only be used upon a value change. It is a programming error otherwise and will cause messages to SPAM the core and GUI. MSG_REG_CHANGED is sent to the GUI and the plugin owning the device that owns the register.

Usage:

```
// Set the register value.
MINSTANCE mInst = 0;
if (mcRegGetHandle(mInst, "core/global/Version", &hReg;) == MERROR_NOERROR) {
 rc = mcRegSetValueString(hReg, "4.0");
}
```

mcRegUnregister

C/C++ Syntax:

```
int mcRegUnregister(
  HMCDEV hDev,
  HMCREG hReg);
```

LUA Syntax:

```
N/A
```

Description: Remove a register from the core.

Parameters:

Parameter	Description
hDev	A HMCDEV specifying the register's parent device.
hReg	A HMCREG specifying the register to remove.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_DEVICE_NOT_FOUND	The **hDev** parameter was 0 or invlaid.
MERROR_REG_NOT_FOUND	The **hReg** parameter was 0 or invlaid.

Notes:
When register is removed all data in that register will be lost.

Usage:

```
MINSTANCE mInst = 0;
HMCREG hReg
char *path = "Sim0/Register1";

mcRegGetHandle(mInst, path, &hReg;);
mcRegUnregister(m_hDev, hReg);
```

Motors

- mcMotorGetAxis
- mcMotorGetInfoStruct
- mcMotorGetMaxAccel
- mcMotorGetMaxVel
- mcMotorGetPos
- mcMotorGetVel
- mcMotorIsHomed
- mcMotorIsStill
- mcMotorRegister
- mcMotorSetHomePos
- mcMotorSetInfoStruct
- mcMotorSetMaxAccel
- mcMotorSetMaxVel
- mcMotorUnregister

mcMotorGetAxis

C/C++ Syntax:

```
int mcMotorGetAxis(
  MINSTANCE mInst,
  int motorId,
  int *axisId)
```

LUA Syntax:

```
axisId, rc = mc.mcMotorGetAxis(
  number mInst,
  number motorId)
```

Description:

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	An integer specifying the motor.
axisId	The address of an interger to receive the motor's mapped axis.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.
MERROR_AXIS_NOT_FOUND	No axis has the specified motor mapped.

Notes:

axisId will be -1 if the function returns MERROR_AXIS_NOT_FOUND.

Usage:

```
// Get the parent axis for motor 0.
MINSTANCE mInst = 0;
int axisId = -1;
int rc = mcMotorGetAxis(mInst, 0, &axisId;);
```

mcMotorGetInfoStruct

C/C++ Syntax:

```
mcMotorGetInfoStruct(
  MINSTANCE mInst,
  int motorId,
  motorinfo_t *minf);
```

LUA Syntax:

```
N/A
```

Description: Retrieve the motor information of the motor in one call.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	The motor id.
minf	The address of a motorinfo_t struct to receive the motor settings.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.
MERROR_INVALID_ARG	**minf** cannot be NULL.

Notes:
Retrieve all the motor settings in one call.
To change any of the settings mcMotorSetInfoStruct() may be called.

```
struct motorinfo {
 double CountsPerUnit;     // Number of encoder counts or steps per unit.
 double MaxVelocity;       // Max velocity of the axis.
 double MaxAcceleration;   // Max rate to accelerate.
 BOOL   Reverse;           // Is the axis reversed?
```

```
 double BacklashAmount;   // The amount of backlash in counts.
 double CurrentVelocity;  // The speed the axis is moving, This could be reporte
 int    CurrentPosition;  // The Current Position (From the motion device).
 BOOL   Homed;            // True if the axis has been homed.
 long   SoftMaxLimit;     // Count for the max travel.
 long   SoftMinLimit;     // Count for the min travel.
 BOOL   CanHome;          // Can this motor home?
 BOOL   Enabled;          // Is this motor enabled?
 long   EnableDelay;      // ms to delay the enable signal for this motor.
 int    AxisId;           // -1 if no axis has mapped this motor.
};
typedef struct motorinfo motorinfo_t;
```

Usage:

```
MINSTANCE mInst = 0;
int m = 2;
motorinfo_t minf;
mcMotorGetInfoStruct(mInst, m, &minf;); // Get data for motor 2.
```

mcMotorGetMaxAccel

C/C++ Syntax:

```
int mcMotorGetMaxAccel(
  MINSTANCE mInst,
  int motorId,
  double *maxAccel);
```

LUA Syntax:

```
maxAccel, rc = mc.mcMotorGetMaxAccel(
  number mInst,
  number motorId)
```

Description: Retrieve the maximum acceleration value for the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	An integer specifying the motor.
maxAccel	The address of a double to receive the maximum acceleration value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
None.

Usage:

```
// Get the max accel for motor 0.
MINSTANCE mInst = 0;
double maxAccel = 0.0;
```

```
int rc = mcMotorGetMaxAccel(mInst, 0, &maxAccel;);
```

mcMotorGetMaxVel

C/C++ Syntax:

```
int mcMotorGetMaxVel(
  MINSTANCE mInst,
  int motorId,
  double *maxVel)
```

LUA Syntax:

```
maxVel, rc = mc.mcMotorGetMaxVel(
  number mInst,
  number motorId)
```

Description: Retrieve the maximum velocity for the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	An integer specifying the motor.
maxVel	The address of a double to receive the maximum velocity value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
None.

Usage:

```
// Get the max vel for motor 0.
MINSTANCE mInst = 0;
double maxVel = 0.0;
```

```
int rc = mcMotorGetMaxVel(mInst, 0, &maxVel;);
```

mcMotorGetPos

C/C++ Syntax:

```
int mcMotorGetPos(
  MINSTANCE mInst,
  int motorId,
  double *val);
```

LUA Syntax:

```
val, rc = mc.mcMotorGetPos(
  number mInst,
  number motorId)
```

Description: Retrieve the position of the given motor in counts (fractional counts supported).

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	An integer specifying the motor.
val	The address of a double to receive the motor position.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.
MERROR_INVALID_ARG	**val** cannot be NULL.

Notes:
If the motor is not setup, **val** will contain zero.

Usage:

```
MINSTANCE mInst = 0;
double Motor1Pos = 0;
// Get the position of the Motor 1.
```

```
mcMotorGetPos(mInst, 1, &Motor1Pos;);
```

mcMotorGetVel

C/C++ Syntax:

```
int mcMotorGetVel(
  MINSTANCE mInst,
  int motor,
  double *velocity);
```

LUA Syntax:

```
velocity, rc = mc.mcMotorGetVel(
  number mInst,
  number motor)
```

Description: Retreive the current velocity of a motor in counts per sec.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	An integer specifying the motor.
velocity	The address of a double to receive the motor velocity.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.
MERROR_INVALID_ARG	**velocity** cannot be NULL.

Notes:
Reports the speed of a motor. The returned value can be fractional counts.
If the motor is not found, a value of zero is returned in velocity.

Usage:

```
MINSTANCE mInst = 0;
int m = MOTOR2;
```

```
double CurrentVel = 0;
mcMotorSetVel(mInst, m, &CurrentVel;); // Get the current speed of motor2.
```

mcMotorIsHomed

C/C++ Syntax:

```
int mcMotorIsHomed(
  MINSTANCE mInst,
  int motorId,
  BOOL *homed);
```

LUA Syntax:

```
homed, rc = mc.mcMotorIsHomed(
  number mInst,
  number motorId)
```

Description: Get the homed state of the motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	The motor id.
homed	The address of a BOOL to receive the homed state of a motor. (TRUE/FALSE)

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.
MERROR_INVALID_ARG	**homed** cannot be NULL.

Notes:
Reports the homed state of the motor. The returned value 1 for homed and 0 for not homed.
All child motors linked to axis must be homed for the axis to report as homed.

Usage:

```
MINSTANCE mInst = 0;
int m = MOTOR2;
int Homed = 0;
mcMotorIsHomed(mInst, m, &Homed;); // Get the homed state of motor2.
```

mcMotorIsStill

C/C++ Syntax:

```
int mcMotorIsStill(
  MINSTANCE mInst,
  int motorId,
  BOOL *still);
```

LUA Syntax:

```
still, rc = mc.mcMotorIsStill(
  number mInst,
  number motorId)
```

Description: Set the motor is not moving from motion Motion plugin that owns this motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorID	The motor id.
still	The address of an BOOL to receive the still state.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.
MERROR_INVALID_ARG	**still** cannot be NULL.

Notes:
Motor has its "isStill" when it is not in motion. **still** == TRUE is no movement, **still** == FALSE motor has movment.

Usage:

```
MINSTANCE mInst = 0;
int m = MOTOR0;
```

```
int still = 0;
mcMotorSetStill(mInst, m, &still;); // Get the state if MOTOR0 is still
```

mcMotorRegister

C/C++ Syntax:

```
mcMotorRegister(
  MINSTANCE mInst,
  int motorId);
```

LUA Syntax:

```
N/A
```

Description: Register a motor to the system (used by motion plugins).

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	The motor id.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**motorId** is out of range.

Notes:
Add motor to the core from a motion device. Motion devices can register max of MC_MAX_MOTORS.

Usage:

```
MINSTANCE mInst = 0;
for (int m = MOTOR0; m < MOTOR4; m++) {
    mcMotorRegister(mInst, m);// Register motor0 - motor2 in the core.
}
```

mcMotorSetHomePos

C/C++ Syntax:

```
int mcMotorSetHomePos(
  MINSTANCE mInst,
  int motorId,
  int count);
```

LUA Syntax:

```
rc = mc.mcMotorSetHomePos(
  number mInst,
  number motorId,
  number count)
```

Description: Set the home position for the motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	The motor id.
count	The positon of the motor away from the home switch

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
Sets the position of the motor away from the home switch. This is used to provide a home offest for the motor.

Usage:

```
MINSTANCE mInst = 0;
int m = MOTOR2;
```

```
int count = 1200;
mcMotorSetHomePos(mInst, m, count); // Set the Home position of motor2.
```

mcMotorSetInfoStruct

C/C++ Syntax:

```
int mcMotorSetInfoStruct(
  MINSTANCE mInst,
  int motoId,
  motorinfo_t *minf);
```

LUA Syntax:

```
N/A
```

Description: Retrieve the motor info struct to get all the settings of the motor in one call.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	The motor id.
minf	The address of a motorinfo_t to receive the motor settings.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.
MERROR_INVALID_ARG	**minf** cannot be NULL.

Notes:
To get cuttent settings mcMotorGetInfoStruct() must be called.

```
struct motorinfo {
 double CountsPerUnit;    // Number of encoder counts or steps per unit.
 double MaxVelocity;      // Max velocity of the axis.
 double MaxAcceleration;  // Max rate to accelerate.
 BOOL   Reverse;          // Is the axis reversed?
 double BacklashAmount;   // The amount of backlash in counts.
```

```
 double CurrentVelocity;  // The speed the axis is moving, This could be reporte
 int    CurrentPosition;  // The Current Position (From the motion device).
 BOOL   Homed;            // True if the axis has been homed.
 long   SoftMaxLimit;     // Count for the max travel.
 long   SoftMinLimit;     // Count for the min travel.
 BOOL   CanHome;          // Can this motor home?
 BOOL   Enabled;          // Is this motor enabled?
 long   EnableDelay;      // ms to delay the enable signal for this motor.
};
typedef struct motorinfo motorinfo_t;
```

Usage:

```
MINSTANCE mInst = 0;
int m = MOTOR2;
motorinfo_t minf;
mcMotorGetInfoStruct(mInst, m, &minf;); // Get data for motor2.
minf.CountsPerUnit /= 2;//Divid motor counts per unit by 2.
mcMotorSetInfoStruct(mInst, m, &minf;); // Set data for motor2.
```

mcMotorSetMaxAccel

C/C++ Syntax:

```
int mcMotorSetMaxAccel(
  MINSTANCE mInst,
  int motorId,
  double maxAccel);
```

LUA Syntax:

```
rc = mc.mcMotorSetMaxAccel(
  number mInst,
  number motorId,
  number maxAccel)
```

Description: Set the maximum acceleration value for the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	An integer specifying the motor.
maxAccel	A double specifying the maximum acceleration value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
None.

Usage:

```
// Set a new accel value for motor 0.
MINSTANCE mInst = 0;
int mcMotorSetMaxAccel(mInst, 0, 75);
```

mcMotorSetMaxVel

C/C++ Syntax:

```
int mcMotorSetMaxVel(
  MINSTANCE mInst,
  int motorId,
  double maxVel);
```

LUA Syntax:

```
rc = mc.mcMotorSetMaxVel(
  number mInst,
  number motorId,
  number maxVel)
```

Description: Set the maximum velocity for the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	An integer specifying the motor.
maxVel	A double specifying the maximum velocity value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
None.

Usage:

```
// Set the maximum velocity for motor 0.
MINSTANCE mInst = 0;
int rc = mcMotorSetMaxVel(mInst, 0, 500);
```

mcMotorUnregister

C/C++ Syntax:

```
int mcMotorUnregister(
  MINSTANCE mInst,
  int motorId);
```

LUA Syntax:

```
N/A
```

Description: Unregister a motor from the core (usually not used.)

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	The motor id.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
Remove motor from the core, this should only be done by the owner of the motor (Motion plugin).

Usage:

```
MINSTANCE mInst = 0;
int m = MOTOR2;
mcMotorUnregister(mInst, m);// Unregister motor2 from the Core.
```

Screw Mapping

Screw mapping is performed with a "teach" API. There are several functions that are simply redundant or allow for setting or retrieving parameters individually. The prefered method for defining a map is as folows:

1. Call mcMotorMapSetDefinition() to define the map.
2. Call mcMotorMapSetPoint() for each defined measurement point.

The motor must be referenced (homed) before starting the mapping process. Similarly, the motor must be homed for the map to be valid! Home is assumed to start at count 0. The length of the screw is defined in counts. mcMotorMapSetPoint() sets the **error**, in counts, for the given point.

Example:

A mill with a 24 inch X axis is to be mapped. Motor 0 is mapped to the X axis. The counts per inch is 20000 and the table is to be measured along it's travel every half inch.

- The length of the screw will be 24 x 20000 for a total of 480000 counts.
- The number of points to be measured would be 48.

```
MINSTANCE mInst = 0;
int rc = mcMotorMapSetDefinition(mInst,
                                 0,      // Motor ID.
                                 480000, // Length or screw in counts.
                                 48);    // Number of measured points.
```

Once the map is defined, the measurment process begins. It is assumed that the table is in the home position (count 0). Move the table to first measurement position moving in the direction **away** from the home switch. A means of precisely measuring the table movement is required. A glass scale or laser can be used. Or by any other means available. But it cannot be stressed enough that whatever method is used, it must be **PRECISE**. Otherwise, the screw map may be less accurate than the error in the screw itself!

In this case, we are measuring the table at every one half inch. Move the table to exactly one half inch and compare the actual motor counts to what the mathematical values should be. In out example, the motor counts should be 10000 (.500" x 20000). If the actual counts required to move the table to the desired position was actually 10004, then we would have an error of 4 counts (positive).

```
rc = mcMotorMapSetPoint(mInst,
                        0,  // Our first measured point.
                        4); // The error, in counts.
```

Repeat the same procedure for every point ending at point 23. All points will need to be measured.

Measuring at random intervals is not supported. The measuring points are evenly spread across the length of the srew. The points in between the measuring points are interpolated to spread the error across the measurment distance. There is no limit to the number of points a map can contain and increasing that number will not slow the system down. About the only impact the number of measuring points will have is in the measurement process itself. More measuring points will of course yield finer resoultion but the user may reach a point of dimisihing returns.

- mcMotorMapGetDefinition
- mcMotorMapGetLength
- mcMotorMapGetPoint
- mcMotorMapGetPointCount
- mcMotorMapGetStart
- mcMotorMapSetDefinition
- mcMotorMapSetLength
- mcMotorMapSetPoint
- mcMotorMapSetPointCount
- mcMotorMapSetStart

mcMotorMapGetDefinition

C/C++ Syntax:

```
int mcMotorMapGetDefinition(
  MINSTANCE mInst,
  int motorId,
  long *lengthCounts,
  long *numPoints);
```

LUA Syntax:

```
lengthCounts, numPoints, rc = mc.mcMotorMapGetDefinition(
  number mInst,
  number motorId)
```

Description: Retrieve the motor screw map definition.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	The motor id.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.
MERROR_INVALID_ARG	**lengthCounts** and **numPoints** are NULL.

Notes:
None.

Usage:

```
//
MINSTANCE mInst = 0;
```

```
long lengthCounts;
long numPoints;
int rc = mcMotorMapGetDefinition(mInst, 0, &lengthCounts;, &numPoints;);
```

mcMotorMapGetLength

C/C++ Syntax:

```
int mcMotorMapGetLength(
  MINSTANCE mInst,
  int motorId,
  int *length);
```

LUA Syntax:

```
length, rc = mcMotorMapGetLength(
  number mInst,
  number motorId)
```

Description: Retrieve the motor screw map length (length of measured screw in counts).

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	The motor id.
length	the address of an integer to receive the screw length in counts.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.
MERROR_INVALID_ARG	**length** is NULL.

Notes:
None.

Usage:

```
// Get the screw length for motor zero.
MINSTANCE mInst = 0;
```

```
int length;
int rc = mcMotorMapGetLength(mInst, 0, &length;);
```

mcMotorMapGetPoint

C/C++ Syntax:

```
int mcMotorMapGetPoint(
  MINSTANCE mInst,
  int motorId,
  int point,
  int *error);
```

LUA Syntax:

```
error, rc = mc.mcMotorMapGetPoint(
  numbser mInst,
  numbser motorId,
  numbser point)
```

Description: Retrieves the screw map error for the given motor and point.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	The motor id.
point	An integer specifying the measurement point number.
error	the address of an integer to receive the screw error (in counts).

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.
MERROR_INVALID_ARG	**error** is NULL.

Notes:
None.

Usage:

```
// Get the screw map error for motor 0, measurement point 10.
MINSTANCE mInst = 0;
int screwErr;
int rc = mcMotorMapGetPoint(mInst, 0, 10, &screwErr;);
```

mcMotorMapGetPointCount

C/C++ Syntax:

```
int mcMotorMapGetPointCount(
  MINSTANCE mInst,
  int motorId,
  int *points);
```

LUA Syntax:

```
points, rc = mc.mcMotorMapGetPointCount(
  number mInst,
  number motorId)
```

Description: Retrieve the number of measurement points in the screw map for the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	The motor id.
points	the address of an integer to receive the number of measurement points.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.
MERROR_INVALID_ARG	**points** is NULL.

Notes:
None.

Usage:

```
// Get the number of measurement points.
MINSTANCE mInst = 0;
```

```
int points;
int rc = mcMotorMapGetPointCount(mInst, 0, &points;);
```

mcMotorMapGetStart

C/C++ Syntax:

```
int mcMotorMapGetStart(
  MINSTANCE mInst,
  int motorId,
  int *startPoint);
```

LUA Syntax:

```
startPoint, rc = mc.mcMotorMapGetStart(
  number mInst,
  number motorId)
```

Description: Retrieve the starting point of the motor screw map for the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	The motor id.
startPoint	The address of an integer to receive the screw map starting point.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.
MERROR_INVALID_ARG	**startPoint** is NULL.

Notes:
None.

Usage:

```
// Get the starting point for the motor 0 screw map.
MINSTANCE mInst = 0;
```

```
int startPoint;
int rc = mcMotorMapGetStart(mInst, 0, &startPoint;);
```

mcMotorMapSetDefinition

C/C++ Syntax:

```
int mcMotorMapSetDefinition(
  MINSTANCE mInst,
  int motorId,
  long lengthCounts,
  long numPoints);
```

LUA Syntax:

```
rc = mc.mcMotorMapSetDefinition(
  number mInst,
  number motorId,
  number lengthCounts,
  number numPoints)
```

Description:

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	The motor id.
lengthCounts	An integer specifying the length of the measured screw in counts.
numPoints	An integer specifying the number of measurement points in the screw map.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:

This is the prefered method of defining a motor screw map.

Usage:

```
// Set the screw map definition for motor 0.
MINSTANCE mInst = 0;
int rc = mcMotorMapSetDefinition(mInst, 0, 20000, 20);
```

mcMotorMapSetLength

C/C++ Syntax:

```
int mcMotorMapSetLength(
  MINSTANCE mInst,
  int motorId,
  int length);
```

LUA Syntax:

```
rc = mc.mcMotorMapSetLength(
  number mInst,
  number motorId,
  number length);
```

Description: Set the length of the screw (in counts) for the screw map of the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	The motor id.
length	An integer specifying the length.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.
MERROR_INVALID_ARG	**length** is less <= 0.

Notes:
None.

Usage:

```
// Set the screw length for the motor 0 screw map.
MINSTANCE mInst = 0;
```

```
int rc = mcMotorMapSetLength(mInst, 0, 20000);
```

mcMotorMapSetPoint

C/C++ Syntax:

```
int mcMotorMapSetPoint(
  MINSTANCE mInst,
  int motorId,
  int point,
  int error);
```

LUA Syntax:

```
rc = mc.mcMotorMapSetPoint(
  number mInst,
  number motorId,
  number point,
  number error);
```

Description: Set the error (in counts) for the screw map point for the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	The motor id.
point	An integer specifying the measurement point number.
error	An integer specifying the screw error.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
None

Usage:

```
// Set the error in motor 0 screw map.
MINSTANCE mInst = 0;
int rc = mcMotorMapSetPoint(mInst, 0, 5, 12);
```

mcMotorMapSetPointCount

C/C++ Syntax:

```
int mcMotorMapSetPointCount(
  MINSTANCE mInst,
  int motorId,
  int points);
```

LUA Syntax:

```
rc = mc.mcMotorMapSetPointCount(
  number mInst,
  number motorId,
  number points);
```

Description: Set the number of measured points for the screw map for the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	The motor id.
points	An integer specifying the number of measured points.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.
MERROR_INVALID_ARG	**points** is <= 0.

Notes:
None.

Usage:

```
// Set the number of measured points in the screw map for motor 0.
MINSTANCE mInst = 0;
```

```
int rc = mcMotorMapSetPointCount(mInst, 0, 20);
```

mcMotorMapSetStart

C/C++ Syntax:

```
int mcMotorMapSetStart(
  MINSTANCE mInst,
  int motorId,
  int start);
```

LUA Syntax:

```
rc = mc.mcMotorMapSetStart(
  number mInst,
  number motorId,
  number start)
```

Description: Set the starting point number of the screw map of the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	The motor id.
start	An integer specifying the starting point number.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
None/

Usage:

```
// Set the starting point number for the motor 0 screw map. (usually 0)
MINSTANCE mInst = 0;
int rc = mcMotorMapSetStart(mInst, 0, 0);
```

Axes

- mcAxisSetOverrideAxis
- mcAxisSetPos
- mcAxisSetSoftlimitEnable
- mcAxisSetSoftlimitMax
- mcAxisSetSoftlimitMin
- mcAxisSetSpindle
- mcAxisSetVel
- mcAxisUnmapMotor
- mcAxisUnmapMotors
- mcAxisUnregister

mcAxisDeref

C/C++ Syntax:

```
int mcAxisDeref(
 MINSTANCE mInst,
 int axisId);
```

LUA Syntax:

```
rc = mc.mcAxisDeref(
 number mInst,
 number axisId)
```

Description: Dereference the specified axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	Axis ID to be dereferenced.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.
MERROR_AXIS_NOT_ENABLED	The axis specified by **axis** is not enabled.

Notes:
The axis will be dereferenced allong with all child motors.

Usage:

```
int mInst = 0;
// Set AXIS0 to deref (could have used X_AXIS).
mcAxisDeref(mInst, AXIS0);
```

mcAxisDerefAll

C/C++ Syntax:

```
int mcAxisDerefAll(
 MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcAxisDerefAll(
 number mInst)
```

Description: Dereference all axes.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
All **ENABLED** axes will be dereferenced.

Usage:

```
int mInst = 0;
// Set all axis to deref.
mcAxisDerefAll(mInst);
```

mcAxisEnable

C/C++ Syntax:

```
int mcAxisEnable(
 MINSTANCE mInst,
 int axisId,
 BOOL enabled);
```

LUA Syntax:

```
rc = mc.mcAxisEnable(
 number mInst,
 number axisId,
 number enabled);
```

Description: Enable or disable the specified axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
enabled	Send true to enable or false to disable axis.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Used to enable or disable axis. Disabling an axis will make motion for the axis impossible.

Usage:

```
int mInst = 0;
// Set Y_AXIS to be disabled.
mcAxisEnable(mInst, Y_AXIS, false);
```

mcAxisGetHomeDir

C/C++ Syntax:

```
int mcAxisGetHomeDir(
  MINSTANCE mInst,
  int axisId,
  int *dir);
```

LUA Syntax:

```
dir, rc = mc.mcAxisGetHomeDir(
  number mInst,
  number axisId)
```

Description: Get the hominig direction for the specified axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
dir	Receives the homing direction. A positive value is home POS. A negative value is home NEG.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
The core provides a container for the axis home direction only. It currently does nothing with it. However, a motion plugin may retrieve the stored direction information and use it.

Usage:

```
int mInst=0;
int dir = 0;
// Get Home offset of the X axis.
```

```
mcAxisGetHomeDir(mInst, X_AXIS, &dir;);
```

mcAxisGetHomeInPlace

C/C++ Syntax:

```
int mcAxisGetHomeInPlace(
  MINSTANCE mInst,
  int axisId,
  BOOL *homeInPlace);
```

LUA Syntax:

```
homeInPlace, rc = mc.mcAxisGetHomeInPlace(
  number mInst,
  number axisId)
```

Description: Determine if the Home In Place flage is set.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	Axis ID.
homeInPlace	Receives the home in place flag value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
The core provides a container for the axis home in place flag only. It currently does nothing with it. However, a motion plugin may retrieve the stored flag and implement a means to home the axis and set machine coordinate 0 at its current location.

Usage:

```
// Get the Home In Place flag for the X axis.
MINSTANCE mInst = 0;
BOOL hip = FALSE;
```

```
mcAxisGetHomeInPlace(mInst, X_AXIS, &hip;);
```

mcAxisGetHomeOffset

C/C++ Syntax:

```
int mcAxisGetHomeOffset(
  MINSTANCE mInst,
  int axisId,
  double *offset);
```

LUA Syntax:

```
offset, rc = mc.mcAxisGetHomeOffset(
  number mInst,
  number axisId)
```

Description: Get the home offset for the specified axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
offset	Receives the home offset.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
The core provides a container for the axis home offset only. It currently does nothing with it. However, a motion plugin may retrieve the stored offset and use it.

Usage:

```
int mInst=0;
int offset = 0;
// Get Home offset of the X axis.
mcAxisGetHomeOrder(mInst, X_AXIS, &offset;);
```

mcAxisGetHomeOrder

C/C++ Syntax:

```
int mcAxisGetHomeOrder(
  MINSTANCE mInst,
  int axisId,
  int *order);
```

LUA Syntax:

```
order, rc = mc.mcAxisGetHomeOrder(
  number mInst,
  number axisId)
```

Description: Get the home order for the specified axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
order	Receives the axis home order number.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Get the home order of the axis so all axis can be homed in the order the user would like.
All axes should be read and then homed in the order requested.

Usage:

```
int mInst=0;
double XAxisPos = 0;
int order = 0;
// Get Home order of the X axis.
mcAxisGetHomeOrder(mInst, X_AXIS, &order);
```

mcAxisGetHomeSpeed

C/C++ Syntax:

```
int mcAxisGetHomeSpeed(
  MINSTANCE mInst,
  int axisId,
  double *percent);
```

LUA Syntax:

```
percent, rc = mc.mcAxisGetHomeSpeed(
  number mInst,
  number axisId)
```

Description:

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
percent	Receives the percentage.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:

The core provides a container for the axis home speed percentage only. It currently does nothing with it. However, a motion plugin may retrieve the stored speed percentage and use it.

Usage:

```
int mInst=0;
int rc = 0;
double XAxisPos = 0;
double percent = 0;
double maxVel = 0
```

```
double homeVel = 0;

// Get home speed percentage of the X axis.
mcAxisGetHomeSpeed(mInst, X_AXIS, &percent;);
mcAxisGetVel(mInst, X_AXIS, &maxVel;);
homeVel = maxVel * percent;
```

mcAxisGetInfoStruct

C/C++ Syntax:

```
int mcAxisGetInfoStruct(
  MINSTANCE mInst,
  int axisId,
  axisinfo_t *ainf);
```

LUA Syntax:

```
N/A
```

Description: Retrieve axis parameters in one call.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	Axis number to be dereffed.
ainf	The address of a axisinfo struct that receives the info for the axis.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
mcAxisGetInfoStruct() is used for quickly retrieving all of the axis parameters. The information is placed in the axisinfo structure. This information can then be modified to update the axis parameters with mcAxisSetInfoStruct().

```
struct axisinfo {
 BOOL OutOfBandAxis;     // Is this an out of band axis?
 BOOL IsStill;           // Set high when the axis is not moving
 BOOL  Jogging;          // Used to tell to jog...
 BOOL  Homing;           // Used to tell the state of the home operation.
 int  Id;                // Axis Id
```

```
 BOOL IsSpindle;          // Does this axis control a spindle?
 BOOL Enabled;            // Is axis enabled?
 BOOL SoftlimitEnabled;   // Softlimits enabled?
 double SoftMaxLimit;     // Count for the max travel.
 double SoftMinLimit;     // Count for the m.
 BOOL VelocityMode;       // Used to make the axis move at a fixed speed
 BOOL BufferJog;          // Buffer all jogs?
 double Pos;              // Position in user units.
 double Mpos;             // Machine position in user units.
 int HomeOrder;           // The order in which to home the axis.
 int HomeDir;             // The direction the axis homes.
 double HomeOffset;       // The offset from the the limits switch.
 double HomeSpeedPercent;// The percentage of the max velocity at which to home.
 BOOL SoftlimitUsed;      // Use Softlimits?
 BOOL HomeInPlace;        // Zero the axis in place when Refed?
 int MotorId[MC_MAX_AXIS_MOTORS]; //child motor ID array.
};
typedef struct axisinfo axisinfo_t;
```

Usage:

```
int mInst = 0;
axisinfo_t ainf;
// Get Y_AXIS info structure.
mcAxisGetInfoStruct(mInst, Y_AXIS, &ainf;);
```

mcAxisGetMachinePos

C/C++ Syntax:

```
int mcAxisGetMachinePos(
 MINSTANCE mInst,
 int axisId,
 double *val);
```

LUA Syntax:

```
val, rc = mc.mcAxisGetMachinePos(
 number mInst,
 number axisId)
```

Description: Get the machine positon of an axis in User units.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
val	Receives the axis machine position in User units.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Get the position of the axis in User units relitive to the home position.

Usage:

```
int mInst=0;
double XAxisMachinePos = 0;
int AxisNumber = X_AXIS;
mcAxisGetMachinePos(mInst, AxisNumber, &XAxisMachinePos;); // Get the position o
```

mcAxisGetMotorId

C/C++ Syntax:

```
int mcAxisGetMotorId(
  MINSTANCE mInst,
  int axis,
  int childId,
  int *motorId);
```

LUA Syntax:

```
motorId, rc = mc.mcAxisGetMotorId(
  number mInst,
  number axis,
  number childId)
```

Description: Get the motor IDs of the motors used on the axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axis	The axis ID from which to get the position.
childId	The axis ID from which to get the position.
motorId	Receives the motor id of the child motor.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.
The motor specified in **child_id** was not found.	

Notes:
Get the motor ID of the child motor. The axis can have many child motors so to get all motors search until MERROR_MOTOR_NOT_FOUND The motor specified by **motorId** was not found. or return is no longer MERROR_NOERROR No Error.

Usage:

```
int mInst=0;
int motorIds[]={-1,-1,-1,-1,-1}
int id;
// Get all the motor ID's for the X axis up to 5 motors.
for(int i = 0; i < 5; i++){
 if (mcAxisGetMotorId(mInst, X_AXIS, i, &id;) ==

MERROR_NOERROR No Error.) {
  motorIds[i] = id;
 }
}
```

mcAxisGetOverrideAxis

C/C++ Syntax:

```
int mcAxisGetOverrideAxis(
  MINSTANCE mInst,
  int axis,
  int *axis1,
  int *axis2,
  int *axis3,
  int *axis4);
```

LUA Syntax:

```
axis1, axis2, axis3, axis4, rc = mc.mcAxisGetOverrideAxis(
  number mInst,
  number axis)
```

Description: Get axis override axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axis	The axis id to apply override axis too.
axis1	Receives the override axis1 ID.
axis2	Receives the override axis2 ID.
axis3	Receives the override axis3 ID.
axis4	Receives the override axis4 ID.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Get the override axis ID for each override axis on the main axis. override_axis_id will return 0 if unused.

Usage:

```
int mInst = 0;
int ora[4];
// Get the override axis for the Z axis.
mcAxisGetOverrideAxis(mInst, ZAXIS, &ora;[0], &ora;[1], &ora;[2], &ora;[3]);
```

mcAxisGetPos

C/C++ Syntax:

```
int mcAxisGetPos(
 MINSTANCE mInst,
 int axisId,
 double *val);
```

LUA Syntax:

```
val, rc = mc.mcAxisGetPos(
 number mInst,
 number axisId)
```

Description: Get the positon of an axis in user units.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID from which to get the position.
val	Receives the axis position in User units.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
The axis position reported is in user units relative to the fixture and tool offsets that are active. This would be the typical value that the user would see in an axis DRO on the GUI of the controller.

Usage:

```
int mInst=0;
double XAxisPos = 0;
int AxisNumber = X_AXIS;
// Get the position of the X axis.
mcAxisGetPos(mInst, AxisNumber, &XAxisPos;);
```

mcAxisGetProbePos

C/C++ Syntax:

```
int mcAxisGetProbePos(
  MINSTANCE mInst,
  int axisId,
  BOOL machinePos,
  double *val);
```

LUA Syntax:

```
val, rc = mc.mcAxisGetProbePos(
  number mInst,
  number axisId,
  number machinePos)
```

Description: Retrieves the position of the last probe strike for the given axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	Axis ID.
machinePos	TRUE for machine coordinates

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.
MERROR_INVALID_ARG	The **val** pointer cannot be NULL.

Notes:

Usage:

```
// Get the last probe strike position for the X axis
MINSTANCE mInst = 0;
double xProbePos = 0;
```

```
mcAxisGetProbePos(mInst, X_AXIS, FALSE, &xProbPos;);
```

mcAxisGetProbePosAll

C/C++ Syntax:

```
int mcAxisGetProbePosAll(
  MINSTANCE mInst,
  BOOL machinePos,
  double *x,
  double *y,
  double *z,
  double *a,
  double *b,
  double *c);
```

LUA Syntax:

```
x, y, z, a, b, c, rc = mc.mcAxisGetProbePosAll(
  number mInst,
  number machinePos)
```

Description: Retrieves the position of the last probe strike for all coordinated axes.

Parameters:

Parameter	Description
mInst	The controller instance.
machinePos	TRUE for machine coordinates
x	The X position.
y	The Y position.
z	The Z position.
a	The A position.
b	The B position.
c	The C position.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

NULL can be specified for any of the parameters x, y, z, a, b, or c.

Usage:

```
// Get the last probe strike position for the X, Y, and Z axes.
MINSTANCE mInst = 0;
double x, y, z;
int rc = mcAxisGetProbePosAll(mInst, FASLE, &x;, &y;, &z;, NULL, NULL, NULL);
if (rc == MERROR_NOERROR) {
 // Process probe positions...
}
```

mcAxisGetScale

C/C++ Syntax:

```
int mcAxisGetScale(
  MINSTANCE mInst,
  int axisId,
  double *scaleVal);
```

LUA Syntax:

```
scaleVal, rc = mc.mcAxisGetProbePos(
  number mInst,
  number axisId)
```

Description: Retrieves the current scaling value for the given axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	Axis ID.
scaleVal	Receives the scale value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.
MERROR_INVALID_ARG	The **scaleVal** pointer cannot be NULL

Notes:
Scale values that are negative produce mirror images.

Usage:

```
// Get the scale value for the X axis
MINSTANCE mInst = 0;
double scale = 0;
mcAxisGetScale(mInst, X_AXIS, &scale;);
```

mcAxisGetSoftlimitEnable

C/C++ Syntax:

```
int mcAxisGetSoftlimitEnable(
   MINSTANCE mInst,
   int axisId,
   int *enable);
```

LUA Syntax:

```
enable, rc = mc.mcAxisGetSoftlimitEnable(
   number mInst,
   number axisId)
```

Description: Get the soft limit enable status.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
enable	Receives the softlimit enable status.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Soft limits can be enabled on a per axis basis. This is the master soft limit enable for the axis. It overrides the soft limit enable set by mcSoftLimitSetState().

Usage:

```
// Get master soft limit enable state for axis 0.
int mInst = 0;
int enable = 0;
mcAxisGetSoftlimitEnable(mInst, 0, &enable;);
```

mcAxisGetSoftlimitMax

C/C++ Syntax:

```
int mcAxisSetSoftlimitMax(
  MINSTANCE mInst,
  int axisId,
  double max);
```

LUA Syntax:

```
rc = mc.mcAxisSetSoftlimitMax(
  number mInst,
  number axisId,
  number max);
```

Description: Set the softlimit maximum for the axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
max	The maximum poisition of the axis in machine coordinates.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Set the softlimit maximum for the axis by drilling down to each motor and setting it max softlimit based on the counts per unit and max sent (max * countsperunit)

Usage:

```
int mInst=0;
```

```
int axis = Y_AXIS;
double max = 20;
mcAxisGetSoftlimitMax( mInst, axis, max);// Set the max distance of the softlimi
```

mcAxisGetSoftlimitMin

C/C++ Syntax:

```
int mcAxisSetSoftlimitMin(
  MINSTANCE mInst,
  int axisId,
  double min);
```

LUA Syntax:

```
rc = mc.mcAxisSetSoftlimitMin(
  number mInst,
  number axisId,
  number min);
```

Description: Set the softlimit minimum for the axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
min	The minimum position of the axis in machine coordinates.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Set the softlimit minimum for the axis by drilling down to each motor and setting it main softlimit based on the counts per unit and min sent (min * countsperunit)

Usage:

```
int mInst=0;
```

```
int axis = Y_AXIS;
double min = 20;
mcAxisGetSoftlimitMin( mInst, axis, min);// Set the min distance of the softlimit
```

mcAxisGetSpindle

C/C++ Syntax:

```
int mcAxisGetSpindle(
  MINSTANCE mInst,
  int axisId,
  bool *spindle);
```

LUA Syntax:

```
spindle, rc = mcAxisGetSpindle(
  number mInst,
  number axisId)
```

Description: Determine if the specified axis is an axis that controls a spindle.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
spindle	Receives the spindle flag value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Only an out of band axis can control a spindle.

Usage:

```
// Find the spindle axis
int mInst = 0;
int axisId;
bool isSpindle = false;
for (axisId = MC_MAX_COORD_AXES; axisId < MC_MAX_AXES; axisId++) {
    mcAxisGetSpindle(mInst, axisId, &isSpindle;);
```

```
    if (isSpindle) {
        // Spindle found!
        break;
    }
}
```

mcAxisGetVel

C/C++ Syntax:

```
int mcAxisGetVel(
 MINSTANCE mInst,
 int axisId,
 double *velocity);
```

LUA Syntax:

```
velocity, rc = mc.mcAxisGetVel(
 number mInst,
 number axisId)
```

Description: Get the velocity of an axis in user units per min.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
velocity	Receives the velocity of the axis in user units per min.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
To get the blended velocity sum the velocity vectors:
blendVel = sqrt(vX*vX + vY*vY + vZ*vZ)

Usage:

```
int mInst=0;
int axis = Y_AXIS;
double CurrentVel = 0;
mcAxisGetVel(mInst, axis, &CurrentVel;); // Get the current speed of the Y axis.
```

mcAxisHome

C/C++ Syntax:

```
int mcAxisHome(
 MINSTANCE mInst,
 int axisId);
```

LUA Syntax:

```
rc = mc.mcAxisHome(
 number mInst,
 number axisId)
```

Description: Used to start an axis homing.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID of which to set the position.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.
MERROR_NOT_NOW	The operation could not be completed at this time.

Notes:
Home the axis, all motors will be homed that are child of the axis.

Usage:

```
int mInst=0;
int AxisNumber = X_AXIS;
// Set the X axis to home.
mcAxisSetPos(mInst, AxisNumber);
```

mcAxisHomeAll

C/C++ Syntax:

```
int mcAxisHomeAll(
 MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcAxisHomeAll(
 number mInst)
```

Description: Used to start an axis homing.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_NOW	The operation could not be completed at this time.

Notes:
Home the all the axes. Only the coordinated axes will be homed.

Usage:

```
int mInst=0;
// Home all coordinated axes.
mcAxisHomeAll(mInst);
```

mcAxisHomeComplete

C/C++ Syntax:

```
int mcAxisHomeComplete(
  MINSTANCE mInst,
  int axisId);
```

LUA Syntax:

```
rc = mc.mcAxisHomeComplete(
  number mInst,
  number axisId);
```

Description: Report that the axis is finished homing.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID which is finished homing.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Reports that the axis is finished homming. This is done by the motion plugin to report when it is finished the homing operation. The motors should be at their home positions and marked as still before this call is made.

Usage:

```
int mInst=0;
int AxisNumber = X_AXIS;
// Mark the X axis home operation as complete.
mcAxisHomeComplete(mInst, AxisNumber);
```

mcAxisHomeCompleteWithStatus

C/C++ Syntax:

```
int mcAxisHomeCompleteWithStatus(
  MINSTANCE mInst,
  int axisId
  BOOL success);
```

LUA Syntax:

```
rc = mc.mcAxisHomeCompleteWithStatus(
  number mInst,
  number axisId
  number success);
```

Description: Report that the axis is finished homing and provide the status of the home operation.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID which is finished homing.
success	The status of the home operation. TRUE or FALSE

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Reports that the axis is finished homming. This is done by the motion plugin to report when it is finished. The motors should be at their home positions and marked as still before this call is made. If the success is set to FALSE, the axis is dereferenced (not homed).

Usage:

```
int mInst=0;
int AxisNumber = X_AXIS;
```

```
// Mark the X axis homing operation as finished and that it completed with succes
mcAxisHomeCompleteWithStatus(mInst, AxisNumber, TRUE);
```

mcAxisIsEnabled

C/C++ Syntax:

```
int mcAxisIsEnabled(
  MINSTANCE mInst,
  int axisId,
  BOOL *enabled);
```

LUA Syntax:

```
enabled, rc = mc.mcAxisIsEnabled(
  number mInst,
  number axisId)
```

Description: Determines if the specified axis is enabled.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
enabled	Receives the axis enable flag value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
This function is primarily used to determine if the user has enabled the axis in the control configuration. It is **NOT** the state of the motor enable signal (OSIG_ENABLEx).

Usage:

```
// Find all enabled axes.
int mInst = 0;
int axisId;
BOOL enabled = FALSE;
for (axisId = 0; axisId < MC_MAX_AXES; axisId++) {
```

```
    enabled = false;
    mcAxisIsEnabled(mInst, axisId, &enabled;);
    if (enabled == TRUE) {
        // Axis is enabled!
    }
}
```

mcAxisIsHomed

C/C++ Syntax:

```
int mcAxisIsHomed(
 MINSTANCE mInst,
 int axisId,
 BOOL *homed);
```

LUA Syntax:

```
homed, rc= mc.mcAxisIsHomed(
  number mInst,
  number axisId)
```

Description: Check to see if axis has been homed.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
homed	Receives the axis homed state (TRUE homed, FALSE not homed).

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Get the homed state of the axis. Axis will only report back as homed if all child motors have been homed.

Usage:

```
int mInst = 0;
BOOL homed = FALSE;
// Get the homed state of Z axis.
```

```
mcAxisIsHomed(mInst, Z_AXIS, &homed;);
if (rc == MERROR_NOERROR && homed == TRUE) {
 // Z is homed.
}
```

mcAxisIsHoming

C/C++ Syntax:

```
int mcAxisIsHomin(
 MINSTANCE mInst,
 int axisId,
 BOOL *homing);
```

LUA Syntax:

```
homing, rc= mc.mcAxisIsHomed(
  number mInst,
  number axisId)
```

Description: Check to see if an axis is in a home operation.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
homing	Receives the axis himing state (TRUE homing, FALSE not homing).

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:

Usage:

```
int mInst = 0;
BOOL homing = FALSE;
// Get the homing state of Z axis.
mcAxisIsHoming(mInst, Z_AXIS, &homing;);
if (rc == MERROR_NOERROR && homing == TRUE) {
 // Z is in a home operation.
```

}

mcAxisIsStill

C/C++ Syntax:

```
int mcAxisIsStill(
 MINSTANCE mInst,
 int axisId,
 BOOL *still);
```

LUA Syntax:

```
still, rc = mc.mcAxisIsStill(
  number mInst,
  number axisId)
```

Description: Report if the axis is still.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
still	Receives a BOOL reflecting the still state.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:

Reports back TRUE if the motor is still and FALSE if it is moving. The data is pulled from the axis' child motors. All of the motors that are mapped to the axis must be still in order for the axis to be marked as still.

Note that an axis can be marked as still even when G code is still processing. A report that the axis is still simply means that the axis is not moving at the point in time the function is called.

Usage:

```
int mInst=0;
int AxisNumber = X_AXIS;
int still = 0;
// Get the is moving state of the axis vlue of 1 is stopped.
mcAxisIsStill(mInst, AxisNumber, &still;);
```

mcAxisMapMotor

C/C++ Syntax:

```
int mcAxisMapMotor(
  MINSTANCE mInst,
  int axisId,
  int motorId);
```

LUA Syntax:

```
rc = mc.mcAxisMapMotor(
  number mInst,
  number axisId,
  number motorId)
```

Description: Map a motor to an axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The ID of the axis to which to map the motor.
motorId	The ID of the motor to map to the specified axis.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
Set the motor to be a child of the axis. Up to six motors can be mapped to an axis. The fist mapped motor is the master motor. Any additional mapped motors are slave motors. Softlimits will be the least common denominator for all the mapped motors.

Usage:

```
int mInst=0;
int AxisNumber = X_AXIS;
int MotorNumber = MOTOR5
// Map motor5 to the X axis.
mcAxisMapMotor(mInst, AxisNumber, MotorNumber);
```

mcAxisRegister

C/C++ Syntax:

```
int mcAxisRegister(
  MINSTANCE mInst,
  int axisId);
```

LUA Syntax:

```
rc = mc.mcAxisRegister(
  number mInst,
  number axisId);
```

Description: Register an axis to the system.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_CREATED	The axis was not created.

Notes:

Register an axis to the system so motors can then be mapped to it. This could be a coordinated axis or an out of band axis.

This function is not used as the core registers axes based on capabilities set by the runtime license.

Usage:

```
int mInst=0;
for (int a = AXIS0; < AXIS4; a++) {
    // Register axis0 - axis2 to the core.
    if( mcAxisRegister(mInst, a) !=
```

```
MERROR_NOERROR No Error.){
          return(false);
     }
}
```

mcAxisRemoveOverrideAxis

C/C++ Syntax:

```
int mcAxisRemoveOverrideAxis(
  MINSTANCE mInst,
  int axisId,
  int overrideId);
```

LUA Syntax:

```
rc = mcAxisRemoveOverrideAxis(
  number mInst,
  number axisId,
  number overrideId)
```

Description: Set axis as an override axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
overrideId	The axis ID of the override axis to be removed.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Remove override axis from the main axis.

Usage:

```
int mInst = 0;
int homed = 0;
// Remove axis 8 from the Z axis.
mcAxisRemoveOverrideAxis( mInst, ZAXIS, AXIS_8);
```

mcAxisSetHomeDir

C/C++ Syntax:

```
int mcAxisSetHomeDir(
  MINSTANCE mInst,
  int axisId,
  int dir);
```

LUA Syntax:

```
rc = mc.mcAxisSetHomeDir(
  number mInst,
  number axisId,
  number dir)
```

Description: Set the specified axis' homing direction.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
dir	The homing direction. -1 (NEG) or 1 (POS)

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
The core provides a container for the axis home direction only. It currently does nothing with it. However, a motion plugin may retrieve the stored direction information and use it.

Usage:

```
// Set axis 0 homing directio to POS.
int mInst = 0;
mcAxisSetHomeDir(mInst, 0, 1);
```

mcAxisSetHomeInPlace

C/C++ Syntax:

```
int mcAxisSetHomeInPlace(
  MINSTANCE mInst,
  int axisId,
  BOOL homeInPlace);
```

LUA Syntax:

```
rc = mc.mcAxisGetHomeInPlace(
  number mInst,
  number axisId,
  number homeInPlace)
```

Description: Set the axis' Home In Place flag.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	Axis ID.
homeInPlace	TRUE or FALSE

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
The core provides a container for the axis home in place flag only. It currently does nothing with it. However, a motion plugin may retrieve the stored flag and implement a means to home the axis and set machine coordinate 0 at its current location.

Usage:

```
// Set the Home In Place flag for the X axis.
MINSTANCE mInst = 0;
```

```
BOOL hip = FALSE;
mcAxisGetHomeInPlace(mInst, X_AXIS, hip);
```

mcAxisSetHomeOffset

C/C++ Syntax:

```
int mcAxisSetHomeOffset(
  MINSTANCE mInst,
  int axisId,
  double offset);
```

LUA Syntax:

```
rc = mcAxisSetHomeOffset(
  number mInst,
  number axisId,
  number offset);
```

Description:

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
offset	Receives the axis home offset..

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:

The core provides a container for the axis home offset only. It currently does nothing with it. However, a motion plugin may retrieve the stored offset and use it.

Usage:

```
// Set the home offset for axis 0.
int mInst = 0;
mcAxisSetHomeOffset(mInst, 0, 1.5 /* inches */);
```

mcAxisSetHomeOrder

C/C++ Syntax:

```
int mcAxisSetHomeOrder(
  MINSTANCE mInst,
  int axisId,
  int order);
```

LUA Syntax:

```
rc = mc.mcAxisSetHomeOrder(
  number mInst,
  number axisId,
  number order)
```

Description: Set the axis home order.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
order	The order in which the axis will be homed (0 is first).

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Set the order the axis is homed in. 0 is the first axis to get homed and more then one axis could have the same home order number.

Usage:

```
int mInst=0;
// Set the order of homming so the Z axis will home first then the X and Y.
mcAxisSetHomeOrder(mInst, Z_AXIS , 0);
mcAxisSetHomeOrder(mInst, X_AXIS , 1);
```

```
mcAxisSetHomeOrder(mInst, Y_AXIS , 1);
```

mcAxisSetHomeSpeed

C/C++ Syntax:

```
int mcAxisSetHomeSpeed(
  MINSTANCE mInst,
  int axis,
  double percent);
```

LUA Syntax:

```
rc = mc.mcAxisSetHomeSpeed(
  number mInst,
  number axis,
  number percent);
```

Description: Set the axis homing velocity as a percentage of the axis maximum velocity.

Parameters:

Parameter	Description
mInst	The controller instance.
axis	The axis ID.
percent	The percentage of the axis maximum velocity expressed in decimal. e.g. 20.5 for 20.5%

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
The core provides a container for the axis home speed percentage only. It currently does nothing with it. However, a motion plugin may retrieve the stored speed percentage and use it.

Usage:

```
// Set the homing speed for axis 0 as a percentage of the max velocity.
int mInst = 0;
```

```
mcAxisSetHomeSpeed(mInst, 0, 20.5 /* percent */);
```

mcAxisSetInfoStruct

C/C++ Syntax:

```
int mcAxisSetInfoStruct(
  MINSTANCE mInst,
  int axisID,
  axisinfo_t *ainf);
```

LUA Syntax:

```
N/A
```

Description: Get axis info struct to get all axis settings in one call.

Parameters:

Parameter	Description
mInst	The controller instance.
axisID	The axis ID.
ainf	The address of a axisinfo_t sruct that receives the info for the axis.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Set all the Axis information in the axis information structure in one call. This is used when all the data needs to be set. Call is best used when paired with mcAxisGetInfoStruct().

```
struct axisinfo {
 bool OutOfBandAxis;
 bool IsStill;           // Set high when the axis is not moving
 int  Jogging;           // Used to tell to jog...
 int  Homing;            // Used to tell the state of the home operation.
 int  Id;                // Axis Id
 bool IsSpindle;   // Does this axis control a spindle?
 bool Enabled;           // Is axis enabled?
```

```
 bool SoftlimitEnabled;  // Softlimits enabled?
 double SoftMaxLimit;    // Count for the max travel.
 double SoftMinLimit;    // Count for the min travel.
 bool VelocityMode;      // Used to make the axis move at a fixed speed
 bool BufferJog;
 double Pos;             // Position in user units.
 double Mpos;            // Machine position in user units.
 int HomeOrder;          // The order in which to home the axis.
};
```

Usage:

```
int mInst = 0;
axisinfo_t ainf;

mcAxisGetInfoStruct(mInst, Y_AXIS, &ainf;);// Get Y_AXIS info structure.
ainf.HomeOrder = 1;// Set Y_AXIS home order to 1.
mcAxisSetInfoStruct(mInst, Y_AXIS, &ainf;);// Set Y_AXIS info structure.
```

mcAxisSetMachinePos

C/C++ Syntax:

```
int mcAxisSetMachinePos(
  MINSTANCE mInst,
  int axis,
  double val);
```

LUA Syntax:

```
rc = mc.mcAxisSetMachinePos(
  number mInst,
  number axis,
  number val)
```

Description:

Parameters:

Parameter	Description
mInst	The controller instance.
axis	The axis ID.
val	The desired axis position to apply a fixture offset.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:

Usage:

```
// Set axis 0 fisxture offset.
int mInst = 0;
mcAxisSetMachinePos(mInst, 0, 0.0 /* set part zero */);
```

mcAxisSetOverrideAxis

C/C++ Syntax:

```
int mcAxisSetOverrideAxis(
  MINSTANCE mInst,
  int axis,
  int overrideId);
```

LUA Syntax:

```
rc = mc.mcAxisSetOverrideAxis(
  number mInst,
  number axis,
  number overrideId)
```

Description: Set axis as an override axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axis	The axis ID to apply override axis too.
overrideId	The axis ID of the of the axis to be used as an override.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Setting an override axis allows an axis to be superimposed on top of the main axis. This can be used as an axis to adjust the main axis on the fly (like Torch Height Control). Up to 4 override axes can be used at the same time.

Usage:

```
int mInst = 0;
int homed = 0;
```

```
// Set the axis 8 to be the override axis for the Z axis.
mcAxisSetOverrideAxis(mInst, ZAXIS, AXIS_8);
```

mcAxisSetPos

C/C++ Syntax:

```
int mcAxisSetPos(
 MINSTANCE mInst,
 int axisId,
 double val);
```

LUA Syntax:

```
rc = mc.mcAxisSetPos(
 number mInst,
 number axisId,
 number val);
```

Description: Set the Position of an axis by changing the fixture offset.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID for which to set the position.
val	The axis position (in user units).

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Set the position of the axis by changing the current Fixture offset. This is not valid for out of band axes.

Usage:

```
int mInst=0;
double XAxisPos = .5;
int AxisNumber = X_AXIS;
// Set the position of the X axis by changing the fixture offset.
```

```
mcAxisSetPos(mInst, AxisNumber, XAxisPos);
```

mcAxisSetSoftlimitEnable

C/C++ Syntax:

```
int mcAxisSetSoftlimitEnable(
   MINSTANCE mInst,
   int axis,
   int enabel);
```

LUA Syntax:

```
rc = mc.mcAxisSetSoftlimitEnable(
   number mInst,
   number axis,
   number enabel)
```

Description: Set the master soft limit enable flag for the specified axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axis	The axis ID.
enable	1 for enable

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Soft limits can be enabled on a per axis basis. This is the master soft limit enable for the axis. It overrides the soft limit enable set by mcSoftLimitSetState().

Usage:

```
// Diable softlimits on axis 0
MINSTANCE mInst = 0;
mcAxisSetSoftlimitEnable(mInst, 0,0);
```

mcAxisSetSoftlimitMax

C/C++ Syntax:

```
int mcAxisSetSoftlimitMax(
  MINSTANCE mInst,
  int axisId,
  double max);
```

LUA Syntax:

```
rc = mc.mcAxisSetSoftlimitMax(
  number mInst,
  number axisId,
  number max);
```

Description: Set the softlimit maximum for the axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
max	The maximum poisition of the axis in machine coordinates.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Set the softlimit maximum for the axis by drilling down to each motor and setting it max softlimit based on the counts per unit and max sent (max * countsperunit)

Usage:

```
int mInst=0;
```

```
int axis = Y_AXIS;
double max = 20;
mcAxisGetSoftlimitMax( mInst, axis, max);// Set the max distance of the softlimi
```

mcAxisSetSoftlimitMin

C/C++ Syntax:

```
int mcAxisSetSoftlimitMin(
  MINSTANCE mInst,
  int axisId,
  double min);
```

LUA Syntax:

```
rc = mc.mcAxisSetSoftlimitMin(
  number mInst,
  number axisId,
  number min);
```

Description: Set the softlimit minimum for the axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
min	The minimum position of the axis in machine coordinates.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Set the softlimit minimum for the axis by drilling down to each motor and setting it main softlimit based on the counts per unit and min sent (min * countsperunit)

Usage:

```
int mInst=0;
```

```
int axis = Y_AXIS;
double min = 20;
mcAxisGetSoftlimitMin( mInst, axis, min);// Set the min distance of the softlimit
```

mcAxisSetSpindle

C/C++ Syntax:

```
int mcAxisSetSpindle(
  MINSTANCE mInst,
  int axisId,
  BOOL spindle);
```

LUA Syntax:

```
rc = mc.mcAxisSetSpindle(
  number mInst,
  number axisId,
  number spindle)
```

Description: Flag or unflag and axis as controlling a spindle.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
spindle	The spindle flag.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:

Setting the current axis as an axis controlling a spindle has the effect of negating this flag for all other axes.
Only out of band axes can be falgged as controlling a spindle.

Usage:

```
// Set axis 6 as a spindle axis.
MINSTANCE mInst = 0;
mcAxisSetSpindle(mInst, 6, true);
```

mcAxisSetVel

C/C++ Syntax:

```
int mcAxisSetVel(
 MINSTANCE mInst,
 int axis,
 double velocity);
```

Description: Not used at this time.

mcAxisUnmapMotor

C/C++ Syntax:

```
int mcAxisUnmapMotor(
  MINSTANCE mInst,
  int axisId,
  int motor);
```

LUA Syntax:

```
rc = mc.mcAxisUnmapMotor(
  number mInst,
  number axisId,
  number motor)
```

Description: Unmap the motor from the axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The ID identifying the axis from which to unmap the motor.
motor	The ID identifying the motor to be unmapped from the aixs.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motor** was not found.

Notes:
Used to remove a motor from an axis based on it's motor id.

Usage:

```
int mInst = 0;
// Remove motor6 from the Y axis.
mcAxisUnmapMotor(mInst, Y_AXIS, MOTOR6);
```

mcAxisUnmapMotors

C/C++ Syntax:

```
int mcAxisUnmapMotors(
  MINSTANCE mInst,
  int axisId);
```

LUA Syntax:

```
rc = mc.mcAxisUnmapMotors(
  number mInst,
  number axisId)
```

Description: Unmap all motors from the axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	Axis number to unmap motor from.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Used to remove all motors from an axis. This is used when an axis needs to have all it's child motors removed and setup with all new motors.

Usage:

```
int mInst = 0;
// Remove all motors from the Y axis.
mcAxisUnmapMotors(mInst, Y_AXIS);
```

mcAxisUnregister

C/C++ Syntax:

```
int mcAxisUnregister(
  MINSTANCE mInst,
  int axisId);
```

LUA Syntax:

```
rc = mc.mcAxisUnregister(
  number mInst,
  number axisId)
```

Description: Unregister an axis from the system.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis id.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:

Usage:

```
int mInst=0;
// Remove AXIS7 from frome the system.
mcAxisUnregister(mInst, AXIS7);
```

Motion

mcMotionClearPlanner

C/C++ Syntax:

```
int mcMotionClearPlanner(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcMotionClearPlanner(
  number mInst)
```

Description: Clears the motion planner of all previously planned moves.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
This function is useful when probing and the probe has struck before the planned moves are exhausted.

Usage:

```
// Clear the planner.
MINSTANCE mInst = 0;
int rc = mcMotionClearPlanner(mInst);
```

mcMotionCyclePlanner

C/C++ Syntax:

```
int mcMotionCyclePlanner(MINSTANCE mInst);
```

LUA Syntax:

```
N/A
```

Description: Cycle the core planner.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
This function is depricated in favor of mcMotionCyclePlannerEx().

Usage:

```
None.
```

mcMotionCyclePlannerEx

C/C++ Syntax:

```
int mcMotionCyclePlannerEx(
  MINSTANCE mInst,
  execution_t *exInfo);
```

LUA Syntax:

```
N/A
```

Description: Cycles the core planner to produce one time slice of movement information.

Parameters:

Parameter	Description
mInst	The controller instance.
exInfo	The address of an execution_t struct that receives the movement information.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**exInfo** is NULL.

Notes:
This is the main function a motion plugin uses to drive motion.

Usage:

```
// Cycle the core planner.
MINSTANCE mInst = 0;
execution_t exInfo;
int rc = mcMotionCyclePlannerEx(mInst, &exInfo;);
// For more information, see the Sim plugin example.
```

mcMotionGetAbsPos

C/C++ Syntax:

```
int mcMotionGetAbsPos(
  MINSTANCE mInst,
  int motorId,
  double *val);
```

LUA Syntax:

```
N/A
```

Description: Retrieves the last planned absolute motor position for the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	An integer specifying the motor.
val	The address of a double to receive the position.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
This function has been depricated in favor of using mcMotionCyclePlannerEx().

Usage:

```
None.
```

mcMotionGetAbsPosFract

C/C++ Syntax:

```
int mcMotionGetAbsPosFract(
  MINSTANCE mInst,
  int motorId,
  double *val);
```

LUA Syntax:

```
N/A
```

Description: Retrieves the last planned absolute motor position for the given motor (with fractions).

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	An integer specifying the motor.
val	The address of a double to receive the position.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
This function has been depricated in favor of using mcMotionCyclePlannerEx().

Usage:

```
None.
```

mcMotionGetBacklashAbs

C/C++ Syntax:

```
int mcMotionGetBacklashAbs(
  MINSTANCE mInst,
  int motorId,
  double *pos);
```

LUA Syntax:

```
N/A
```

Description: Retrieves the last planned backlast amount for the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	An integer specifying the motor.
pos	The address of a double to receive the position.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
This function has been depricated in favor of using mcMotionCyclePlannerEx().

Usage:

```
None.
```

mcMotionGetBacklashInc

C/C++ Syntax:

```
int mcMotionGetBacklashInc(
  MINSTANCE mInst,
  int motorId,
  double *pos);
```

LUA Syntax:

```
N/A
```

Description: Retrieves the last planned backlast amount for the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	An integer specifying the motor.
pos	The address of a double to receive the position.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
This function has been depricated in favor of using mcMotionCyclePlannerEx().

Usage:

```
None.
```

mcMotionGetIncPos

C/C++ Syntax:

```
int mcMotionGetIncPos(
  MINSTANCE mInst,
  int motorId,
  double *val);
```

LUA Syntax:

```
N/A
```

Description: Retrieves the last planned incremental motor position for the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	An integer specifying the motor.
val	The address of a double to receive the position.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
This function has been depricated in favor of using mcMotionCyclePlannerEx().

Usage:

```
None.
```

mcMotionGetMoveID

C/C++ Syntax:

```
int mcMotionGetMoveID(
  MINSTANCE mInst,
  int motorId,
  long *val);
```

LUA Syntax:

```
N/A
```

Description: Retrieves the last planned move ID for the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	An integer specifying the motor.
val	The address of a long to receive the value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
This function has been depricated in favor of using mcMotionCyclePlannerEx().

Usage:

```
None.
```

mcMotionGetPos

C/C++ Syntax:

```
int mcMotionGetPos(
  MINSTANCE mInst,
  int motorId,
  double *pos);
```

LUA Syntax:

```
pos, rc = mc.mcMotionGetPos(
  number mInst,
  number motorId)
```

Description: Retrieves the current motor pos in counts.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	An integer specifying the motor.
pos	The address of a double to receive the position.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
None.

Usage:

```
// Get the motor 0 position.
MINSTANCE mInst = 0;
double pos;
int rc = mcMotionGetPos(mInst, 0, &pos;);
```

mcMotionGetProbeParams

C/C++ Syntax:

```
int mcMotionGetProbeParams(
  MINSTANCE mInst,
  probe_t *probeInfo);
```

LUA Syntax:

```
N/A
```

Description: Get the probing parameters from the core.

Parameters:

Parameter	Description
mInst	The controller instance.
probeInfo	The address of a prob_t struct to receive the probing parameters.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
A motion device registered with DEV_TYPE_PROBE2 must be defined in order to use this function.

Usage:

```
// Get the probing parameters from the core.
MINSTANCE mInst = 0;
probe_t pInfo;
int mcMotionGetProbeParams(mInst, &pInfo;);
```

mcMotionGetRigidTapParams

C/C++ Syntax:

```
int mcMotionGetRigidTapParams(
  MINSTANCE mInst,
  tap_t *tapInfo);
```

LUA Syntax:

```
N/A
```

Description: Retrieve the tapping parameters from the core.

Parameters:

Parameter	Description
mInst	The controller instance.
tapInfo	The address of a tap_t struct to receive the tapping parameters.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
A motion device registered with DEV_TYPE_TAP2 must be defined in order to use this function.

Usage:

```
// Get the tpping parameters from the core.
MINSTANCE mInst = 0;
tap_t tapInfo;
int rc = mcMotionGetRigidTapParams(mInst, &tapInfo;);
```

mcMotionGetSyncOutput

C/C++ Syntax:

```
int mcMotionGetSyncOutput(
  MINSTANCE mInst,
  int outputQueue,
  HMCIO *hIo,
  BOOL *state);
```

LUA Syntax:

```
N/A
```

Description: Gets a movement coordinated output from the core.

Parameters:

Parameter	Description
mInst	The controller instance.
outputQueue	An integer specifying which output queue.
hIo	The address of a HMCIO variable to receive the I/O handle associated with the output.
state	Te address of a BOOL to receive the desired state of the output.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Test the ex.exOutputQueue flag to see if there are any coordinated output to be set/cleared. The **outputQueue** parameter comes from ex.ex.exOutputQueue.

Usage:

```
// Retrieve the coordinated outputs/
MINSTANCE mInst = 0;
execution_t ex;
mcMotionCyclePlannerEx(mInst, &ex;);
```

```
if (ex.exOutputQueue != EX_NONE) {
 HMCIO hIo;
 BOOL state;
 while (mcMotionGetSyncOutput(m_cid, ex.exOutputQueue, &hIo;, &state;) == MERROR_
  // Pair setting the output along with the moves in (execution_t)ex.
 }
}
```

mcMotionGetThreadParams

C/C++ Syntax:

```
int mcMotionGetThreadParams(
  MINSTANCE mInst,
  thread_t *threadInfo);
```

LUA Syntax:

Description:

Parameters:

Parameter	Description
mInst	The controller instance.
threadInfo	The address of a thread_t struct to receive the threading parameters.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_IMPLEMENTED	The motion device does not support DEV_TYPE_THREAD2 type threading.

Notes:
A motion device registered with DEV_TYPE_THREAD2 is required.

Usage:

```
//
MINSTANCE mInst = 0;
thread_t threadInfo;
int mcMotionGetThreadParams(mInst, &threadInfo;);
```

mcMotionGetThreadingRate

C/C++ Syntax:

```
int mcMotionGetThreadingRate(
  MINSTANCE mInst,
  double *ratio);
```

LUA Syntax:

```
N/A
```

Description: Get the current threading ration from the core.

Parameters:

Parameter	Description
mInst	The controller instance.
ratio	The address to a double to receive the threading ratio.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

Usage:

```
// Get the current threading rate from the core.
MINSTANCE mInst = 0;
double ratio;
int rc = mcMotionGetThreadingRate(mInst, :);
```

mcMotionGetVel

C/C++ Syntax:

```
int mcMotionGetVel(MINSTANCE mInst, int motorId, double *velocity);
```

LUA Syntax:

```
velocity, rc = mc.mcMotionGetVel(
  number mInst,
  number motorId)
```

Description: Retrieve the current motor velocity in counts per second squared.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	An integer specifying the motor ID.
velocity. The address of a double to receive the motor velocity.	

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.
MERROR_INVALID_ARG	**velocity** is NULL or **motorId** is less than 0.

Notes:
None.

Usage:

```
// Get the motor velocity for motor 1
MINSTANCE mInst = 0;
double vel;
int rc = mcMotionGetVel(mInst, 1, &vel;);
```

mcMotionSetCycleTime

C/C++ Syntax:

```
int mcMotionSetCycleTime(
  MINSTANCE mInst,
  double secs);
```

LUA Syntax:

```
N/A
```

Description: Set the time slice of a cycle.

Parameters:

Parameter	Description
mInst	The controller instance.
secs	A double specifying the cycle time slice.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Decimal values are accepted. e.g. .001 is equal to 1 millisecond.

Usage:

```
// Set the cycle time slice.
MINSTANCE mInst = 0;
int mcMotionSetCycleTime(mInst, .001);
```

mcMotionSetMoveID

C/C++ Syntax:

```
int mcMotionSetMoveID(
  MINSTANCE mInst,
  int id);
```

LUA Syntax:

```
N/A
```

Description: Set the time slice of a cycle.

Parameters:

Parameter	Description
mInst	The controller instance.
id	Set the currently executing movement ID.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Ideally, the motion control device should report the currently executing movement.

Usage:

```
// Set the currently executing movement ID.
MINSTANCE mInst = 0;
int moveId = 10001; // Should be obtained from the motion controller.
int mcMotionSetMoveID(mInst, moveId);
```

mcMotionSetPos

C/C++ Syntax:

```
int mcMotionSetPos(
  MINSTANCE mInst,
  int motorId
  double val);
```

LUA Syntax:

```
N/A
```

Description: Report the position in counts to the core for the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
MotorId	An integer specifying the motor.
val	A double specifying the motor position in counts.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
The motion control device should report the motor positions to the core so that the core can synchronize events based on the current position.

Usage:

```
// Set the motor position for motor 1.
MINSTANCE mInst = 0;
motorCounts = 324255; // Should be retrieved from the motion controller.
int mcMotionSetPos(mInst, 1, motorCounts);
```

mcMotionSetProbeComplete

C/C++ Syntax:

```
int mcMotionSetProbeComplete(
  MINSTANCE mInst);
```

LUA Syntax:

```
N/A
```

Description: Inform the core that a probe operation is complete.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Calling this function assumes that a probe stike has happened and that the probed positions have been reproted.

Usage:

```
// Complete a probe operation.
MINSTANCE mInst = 0;
int rc = mcMotionSetProbeComplete(mInst);
```

mcMotionSetProbePos

C/C++ Syntax:

```
int mcMotionSetProbePos(
  MINSTANCE mInst,
  int motorId,
  double val);
```

LUA Syntax:

```
N/A
```

Description: Report the probed position for the given motor to the core.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	An integer specifying the motor.
val	A double specifying the probed motor position in counts.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
When the probe strikes, the position needs to be reported to the core.

Usage:

```
// Set the probed position for motor 0.
MINSTANCE mInst = 0;
double probedPos = 2314134; // Should be reteived from motion controller latch re
int rc = mcMotionSetProbePos(mInst, 0, double val);
```

mcMotionSetStill

C/C++ Syntax:

```
int mcMotionSetStill(
  MINSTANCE mInst,
  int motorId);
```

LUA Syntax:

```
N/A
```

Description: Report when the given motor is still.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	An integer specifying the motor.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
The core will request a stop report in order to synchronize events with the end of movement. This is a very important thing to get correct in a motion plugin. This is how the core will know to stop waiting on M code or any other G code that breaks the CV chain.

Usage:

```
// Motor stop report example.
MINSTANCE mInst = 0;
execution_t ex;
mcMotionCyclePlannerEx(m_cid, &ex;);
switch(ex.exType) {
case EX_STOP_REQ:
 // See if we need to report when the motors are still.
```

```
 for (i = 0; i < 8; i++) {
  if (ex.exMotors[i].reportStopped == TRUE) {
   // Report when this motor has completed all previous moves!
  }
 }
 break;
 ...
}
```

mcMotionSetThreadingRate

C/C++ Syntax:

```
int mcMotionSetThreadingRate(
  MINSTANCE mInst,
  double ratio);
```

LUA Syntax:

```
N/A
```

Description: Set the threading skew ratio.

Parameters:

Parameter	Description
mInst	The controller instance.
ratio	A double specifying the skew ratio.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Set the threading ratio.
MINSTANCE mInst = 0;
int rc = mcMotionSetThreadingRate(mInst, 1.2);
```

mcMotionSetVel

C/C++ Syntax:

```
int mcMotionSetVel(
  MINSTANCE mInst,
  int motorId,
  double velocity);
```

LUA Syntax:

```
N/A
```

Description: Report the velocity (in counts per second squared) to the core for the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	An integer specifying the motor.
velocity	A double specifying the velocity in counts per second squared.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
Motion plugins use this function to report motor velocities to the core.

Usage:

```
// Report the velocity for motor 0.
MINSTANCE mInst = 0;
double vel = 2342420;// Should be obtained from the motion controller.
int mcMotionSetVel(mInst, 0, vel);
```

mcMotionSync

C/C++ Syntax:

```
int mcMotionSync(
  MINSTANCE mInst);
```

LUA Syntax:

```
N/A
```

Description: Synchronize the core planners with the last reported motor positions.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
If movement has been done outside of direction from the core, the planners must be synced to the new motor positons.

Usage:

```
// Synch core planner positions with the last reported motor positions.
MINSTANCE mInst = 0;
int rc = mcMotionSync(mInst);
```

mcMotionThreadComplete

C/C++ Syntax:

```
int mcMotionThreadComplete(
  MINSTANCE mInst);
```

LUA Syntax:

```
N/A
```

Description: Inform the core that the threading operation is complete.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
If the motion controller is profiling all of the threading moves, the core must be informed when the threading operation is complete.

Usage:

```
// Report that the threading op is complete.
MINSTANCE mInst = 0;
int rc = mcMotionThreadComplete(mInst);
```

Operation

- mcCntlCycleStart
- mcCntlCycleStop
- mcCntlDryRunToLine
- mcCntlEStop
- mcCntlFeedHold
- mcCntlFeedHoldState
- mcCntlGetBlockDelete
- mcCntlGetFRO
- mcCntlGetOptionalStop
- mcCntlGetRRO
- mcCntlGetSingleBlock
- mcCntlGotoZero
- mcCntlIsInCycle
- mcCntlReset
- mcCntlSetBlockDelete
- mcCntlSetFRO
- mcCntlSetOptionalStop
- mcCntlSetRRO
- mcCntlSetResetCodes
- mcCntlSetSingleBlock
- mcCntlStartMotionDev
- mcCntlStopMotionDev
- mcCntlToolChangeManual
- mcCntlWaitOnCycleStart

mcCntlCycleStart

C/C++ Syntax:

```
int mcCntlCycleStart(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcCntlCycleStart(
  number mInst)
```

Description: Start the Gcode file running.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_NOW	The operation could not be completed at this time.

Notes:
Is also used to restart after a feedhold.

Usage:

```
int mInst=0;
// Start controller 0's Gcode file.
mcCntlCycleStart(mInst);
```

mcCntlCycleStop

C/C++ Syntax:

```
int mcCntlCycleStop(
 MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcCntlCycleStop(
  number mInst)
```

Description: Stop a Gcode file that is running.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_NOW	The operation could not be completed at this time.

Notes:
Feedhold needs to be used to pause the file. CMD_STOP_EXECUTION is sent to all plugins to tell the the Stop event has been requested.

Usage:

```
int mInst=0;
// Stop controller 0's Gcode file.
mcCntlCycleStop(mInst);
```

mcCntlDryRunToLine

C/C++ Syntax:

```
int mcCntlDryRunToLine(
  MINSTANCE mInst,
  int line);
```

LUA Syntax:

```
rc = mc.mcCntlDryRunToLine(
  number mInst,
  number line)
```

Description: Process G code forward to a given line without motion.

Parameters:

Parameter	Description
mInst	The controller instance.
line	An integer specifying the G code line number.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_ENABLED	The control is not enabled.
MERROR_NOT_NOW	The operation could not be completed at this time.
It was not possible to start the dry run at this time.	

Notes:
Once the dry run is started, the function returns immediately. To wait on the dry run, you must check the state of the control.

Usage:

```
// Dry run to line 38
MINSTANCE mInst = 0;
```

```
int rc = mcCntlDryRunToLine(mInst, 38);
```

mcCntlEStop

C/C++ Syntax:

```
int mcCntlEStop(
 MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcCntlEStop(
 number mInst)
```

Description: Disables the control and optionally dereferences all axes.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
This function is not a subsitute for a hard wired e-stop circuit! This is a convenience function that is designed to put the contorol into a default state to be used after the machine has already been stopped by the real e-stop control circuit.

Usage:

```
// Put the control in the default state when e-stop is asserted.
MINSTANCE mInst = 0;
int rc = mcCntlEStop(mInst);
```

mcCntlFeedHold

C/C++ Syntax:

```
int mcCntlFeedHold(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcCntlFeedHold(
  number mInst)
```

Description: Pause the motion of a Gcode block.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_NOW	The operation could not be completed at this time.

Notes:
Will not effect out of band axis or jogging.

Usage:

```
int mInst = 0;
int rc;
rc = mcCntlFeedHold(mInst); // Pause the motion of controller 0.
if (rc == MERROR_NOERROR) {
 // Feed hold was successful.
}
```

mcCntlFeedHoldState

C/C++ Syntax:

```
int mcCntlFeedHoldState(
  MINSTANCE mInst,
  BOOL *InFeedHold);
```

LUA Syntax:

```
InFeedHold, rc = mc.mcCntlFeedHoldState(
    number mInst)
```

Description: Get the state of Feedhold.

Parameters:

Parameter	Description
mInst	The controller instance.
InFeedHold	The address of a BOOL to receive the status of Feedhold (TRUE or FALSE).

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Not used at this time.

Usage:

```
int mInst = 0;
int rc;
BOOL fhState;
rc = mcCntlFeedHoldState(mInst, &fhState;); // Pause the motion of controller 0.
if (rc == MERROR_NOERROR && fhState == TRUE) {
 // The control is in Feedhold.
}
```

mcCntlGetBlockDelete

C/C++ Syntax:

```
int mcCntlGetBlockDelete(
  MINSTANCE mInst,
  int deleteId,
  BOOL *val);
```

LUA Syntax:

```
val, rc = mc.mcCntlGetBlockDelete(
  number mInst,
  number deleteId)
```

Description:

Parameters:

Parameter	Description
mInst	The controller instance.
deleteId	An integer specifying the block delete index.
val	The address of a BOOL to revceive the block delete state. (TRUE is block delete on

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

Up to 10 block deletes can be used.

Usage:

```
// Is block delete 0 on?
MINSTANCE mInst = 0;
BOOL val;
int rc = mcCntlGetBlockDelete(mInst, 0, &val;);
if (rc == MERROR_NOERROR) {
 if (val == TRUE) {
  // Is block delete 0 is on!
 } else {
```

```
  // Is block delete 0 is off!
 }
}
```

mcCntlGetFRO

C/C++ Syntax:

```
int mcCntlGetFRO(
  MINSTANCE mInst,
  double *percent);
```

LUA Syntax:

```
percent, rc = mc.mcCntlGetFRO(
  number mInst)
```

Description: Get the current Feed Rate Override percentage.

Parameters:

Parameter	Description
mInst	The controller instance.
percent	The Address of a double to receive the feed rate override percentage.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Get the current FRO
MINSTANCE mInst = 0;
double fro;
int rc = mcCntlGetFRO(mInst, &fro;);
if (rc == MERROR_NOERROR) {
 // Success!
}
```

mcCntlGetOptionalStop

C/C++ Syntax:

```
int mcCntlGetOptionalStop(
  MINSTANCE mInst,
  BOOL *stop);
```

LUA Syntax:

```
stop, rc = mc.mcCntlGetOptionalStop(
  number mInst)
```

Description: Retrieve the state of optional stop.

Parameters:

Parameter	Description
mInst	The controller instance.
stop	The address of a BOOL to contain the optional stop state.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**stop** is NULL.

Notes:
None.

Usage:

```
// See if optional stop is in effect.
MINSTANCE mInst = 0;
BOOL opStop = FALSE;
int rc = mcCntlGetOptionalStop(mInst, &opStop;);
if (rc == MERROR_NOERROR) {
 if (opStop == TRUE) {
  // Optional Stop is in effect!
 } else {
  // Optional Stop is not in effect!
 }
```

}

mcCntlGetRRO

C/C++ Syntax:

```
int mcCntlGetRRO(
  MINSTANCE mInst,
  double *percent);
```

LUA Syntax:

```
percent, rc = mc.mcCntlGetRRO(
  number mInst)
```

Description: Get the current Feed Rate Override percentage.

Parameters:

Parameter	Description
mInst	The controller instance.
percent	The Address of a double to receive the rapid rate override percentage.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Get the current RRO
MINSTANCE mInst = 0;
double rro;
int rc = mcCntlGetRRO(mInst, &rro;);
if (rc == MERROR_NOERROR) {
 // Success!
}
```

mcCntlGetSingleBlock

C/C++ Syntax:

```
int mcCntlGetSingleBlock(
  MINSTANCE mInst,
  BOOL *sbState);
```

LUA Syntax:

```
sbState, rc = mc.mcCntlGetSingleBlock(
  number mInst)
```

Description: Get the state of Single block.

Parameters:

Parameter	Description
mInst	The controller instance.
sbState	The address of a BOOL to receive the state of single block. (MC_ON/TRUE, MC_OFF/FALSE)

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**sbState** is NULL.

Notes:
Used to display the state of single block to the user.

Usage:

```
// Get the state of single block.
int mInst=0;
BOOL IsOn = FASLE;
mcCntlGetSingleBlock(mInst, &IsOn;);
```

mcCntlGotoZero

C/C++ Syntax:

```
int mcCntlGotoZero(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcCntlGotoZero(
  number mInst)
```

Description: Move the X,Y,A,B,C and then the Z axis to zero of the current fixture offset.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_NOW	The operation could not be completed at this time.

Notes:
This can also be done by sending in a G code move.

Usage:

```
int mInst = 0;
mcCntlGotoZero(mInst); // Move controller instance 0 to x,y,z,a,b,c zero.
```

mcCntlIsInCycle

C/C++ Syntax:

```
int mcCntlIsInCycle(
  MINSTANCE mInst,
  BOOL *cycle);
```

LUA Syntax:

```
cycle, rc = mc.mcCntlIsInCycle(
  number mInst)
```

Description: Used to see if a G code file is running.

Parameters:

Parameter	Description
mInst	The controller instance.
cycle	The address of a BOOL to receive the in cycle state (TRUE, FALSE).

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**cycle** is NULL.

Notes:
This call is useful for showing the user if the control is processing a G code file.

Usage:

```
int mInst = 0;
BOOL InCycle = FALSE;
bool RunningFile = false;

int rc = mcCntlIsInCycle(mInst, &InCycle;); // See if a G code file is running.
if(rc == MERROR_NOERROR && InCycle == TRUE){
    RunningFile = true;
}
```

mcCntlReset

C/C++ Syntax:

```
int mcCntlReset(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcCntlReset(
  number mInst)
```

Description: Reset the controller instance to a known state.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_NOW	The operation could not be completed at this time.

Notes:
A MSG_RESET_CNTL notification is sent to the GUI and plugins.

Usage:

```
MINSTANCE mInst = 0;
mcCntlReset(mInst); // Reset controller instance 0.
```

mcCntlSetBlockDelete

C/C++ Syntax:

```
int mcCntlSetBlockDelete(
  MINSTANCE mInst,
  int deleteID,
  BOOL enabled);
```

LUA Syntax:

```
rc = mc.mcCntlSetBlockDelete(
  number mInst,
  number deleteID,
  number enabled)
```

Description: Enable or Disable a block delete level.

Parameters:

Parameter	Description
mInst	The controller instance.
deleteId	The block delete index level.
enabled	A BOOL specifying if the lock delete level is enabled. (TRUE/FALSE)

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**deleteID** is out of range.

Notes:
There are currently 10 block delete levels (0-9). If any block delete level is enabled, the OSIG_BLOCK_DELETE output signal is asserted.

Usage:

```
MINSTANCE mInst = 0;
// Enable block delete level 0.
mcCntlSetBlockDelete(mInst, 0, TRUE);
```

mcCntlSetFRO

C/C++ Syntax:

```
int mcCntlSetFRO(
  MINSTANCE mInst,
  double percent);
```

LUA Syntax:

```
rc = mc.mcCntlSetFRO(
  number mInst,
  number percent);
```

Description: Set the feed rate override by the percentage of the coded feed rate.

Parameters:

Parameter	Description
mInst	The controller instance.
percent	A double specifying the feed rate override percent.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

Usage:

```
// Set feed rate override to 150%
MINSTANCE mInst = 0;
int rc = mcCntlSetFRO(mInst, 150.0);
```

mcCntlSetOptionalStop

C/C++ Syntax:

```
int mcCntlSetOptionalStop(
  MINSTANCE mInst,
  BOOL enable);
```

LUA Syntax:

```
rc = mc.mcCntlSetOptionalStop(
  number mInst
  number enable)
```

Description: Set the state of optional stop.

Parameters:

Parameter	Description
mInst	The controller instance.
enable	A BOOL specifying the state of optional stop. (TRUE/FALSE)

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
The output signal OSIG_OPT_STOP reflects the state of optional stop.

Usage:

```
// Enable optional stop.
MINSTANCE mInst = 0;
int rc = mcCntlSetOptionalStop(mInst, TRUE);
```

mcCntlSetRRO

C/C++ Syntax:

```
int mcCntlSetRRO(
  MINSTANCE mInst,
  double percent);
```

LUA Syntax:

```
rc = mc.mcCntlSetRRO(
  number mInst,
  number percent);
```

Description: Set the rapid rate override by the percentage of maxiumum velocity.

Parameters:

Parameter	Description
mInst	The controller instance.
percent	A double specifying the rapid rate override percentage.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
The maximum velocity is capped by the motor tuning. Therefore the percentage should be less than 100%.

Usage:

```
// Set feed rapid override to 50%
MINSTANCE mInst = 0;
int rc = mcCntlSetRRO(mInst, 50.0);
```

mcCntlSetResetCodes

C/C++ Syntax:

```
int mcCntlSetResetCodes(
  MINSTANCE mInst,
  const char *resetCodes);
```

LUA Syntax:

```
rc = mc.mcCntlSetResetCodes(
  number mInst,
  string resetCodes)
```

Description: Set the G code that is used when mcCntlReset() is called.

Parameters:

Parameter	Description
mInst	The controller instance.
resetCodes	A string buffer containing the G code.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Each block needs to be separated with new line characters (\n).

Usage:

```
// Set the G code used to reset the controller instance 0.
MINSTANCE mInst = 0;
int rc = mcCntlSetResetCodes(mInst, "G40 G20 G90 G52 X0 Y0 Z0\nG92.1 G69");
```

mcCntlSetSingleBlock

C/C++ Syntax:

```
int mcCntlSetSingleBlock(
  MINSTANCE mInst,
  BOOL enable);
```

Description: Set the state of single block mode.

Parameters:

Parameter	Description
mInst	The controller instance.
enable	A BOOL specifying the state of single block mode. (TRUE/FALSE).

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Single block runs one line of G code at a time and is most often used to debug G code programs.

Usage:

```
// Enable single block mode.
MINSTANCE mInst = 0;
mcCntlSetSingleBlock(mInst, TRUE);
```

mcCntlStartMotionDev

C/C++ Syntax:

```
int mcCntlStartMotionDev(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcCntlStartMotionDev(
  number mInst)
```

Description: Starts the selected motion device.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Used to start the selected motion device. This is normally a function used by a GUI where a motion device selection dialog is presented to the user.

Usage:

```
// Start the selected motion device.
MINSTANCE mInst = 0;
int rc = mcCntlStartMotionDev(mInst);
```

mcCntlStopMotionDev

C/C++ Syntax:

```
int mcCntlStopMotionDev(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcCntlStopMotionDev(
  number mInst);
```

Description: Stop the selected motion device.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Used to stop the selected motion device. This is normally a function used by a GUI where a motion device selection dialog is presented to the user.

Usage:

```
// Stop the selected motion device.
MINSTANCE mInst = 0;
int rc = mcCntlStopMotionDev(mInst);
```

mcCntlToolChangeManual

C/C++ Syntax:

```
int mcCntlToolChangeManual(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcCntlToolChangeManual(
  number mInst)
```

Description: A convenience function used in M6 tool change macros. It will prompt the user to change the tool and wait for a cycle start.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_NOW	The operation could not be completed at this time.

Notes:
This function is primarily used in a scripting environment.

Usage:

```
-- LUA example
local inst = mc.mcGetInstance();
local rc = mc.mcCntlToolChangeManual(inst);
```

mcCntlWaitOnCycleStart

C/C++ Syntax:

```
int mcCntlWaitOnCycleStart(
  MINSTANCE mInst,
  const char *msg,
  int timeOutMs);
```

LUA Syntax:

```
rc = mc.mcCntlWaitOnCycleStart(
  number mInst,
  string msg,
  number timeOutMs);
```

Description: A convenience function used in macros. It will prompt the user with the given message and optionally use a timeout.

Parameters:

Parameter	Description
mInst	The controller instance.
msg	A string buffer for the prompt message.
timeOutMs	A timeout value in milliseconds.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_NOW	The operation could not be completed at this time.

Notes:
This function is primarily used in a scripting environment.

Usage:

```
-- LUA example
local inst = mc.mcGetInstance();
local rc = mc.mcCntlWaitOnCycleStart(inst, "Please press Cycle Start.", 1000);
```

GUI

mcCntlCleanup

C/C++ Syntax:

```
int mcCntlCleanup(MINSTANCE mInst);
```

LUA Syntax:

```
N/A
```

Description: Cleanup the core instance before shutdown.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
None.
// Cleanup core instance 0
MINSTANCE mInst = 0;
int rc = mcCntlCleanup(mInst);
```

mcCntlInit

C/C++ Syntax:

```
MINSTANCE mcCntlInit(
  const char *ProfileName,
  int ControllerID);
```

LUA Syntax:

```
N/A
```

Description: Init the instance specified by **ControllerID** with the profile specified by **ProfileName**.

Parameters:

Parameter	Description
ProfileName	Profile that is to be loaded for the controller instance.
ControllerID	The controller instance to start. (must start with zero)

Returns:

Return Code	Description
The controller instance.	0 to 5
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MACHINIT_INVALID_ARG	**ProfileName** is NULL.
MACHINIT_CONTROLLER_INUSE	The controller instance is in use, try the next one.

Notes:
Only GUIs will use this API function. It must be called at the start of the app to initialize at least one controller instance. All other API calls will fail with MERROR_INVALID_INSTANCE The **mInst** parameter was out of range. until an instance is initialized.

Usage:

```
MINSTANCE mInst;
mInst = mcCntlInit("Mach4Mill", mInst);
if (mInst >= 0) {
 // A valid instance has been initialized!
}
```

mcFixtureLoadFile

C/C++ Syntax:

```
int mcFixtureLoadFile(
  MINSTANCE mInst,
  const char *FileToLoad);
```

LUA Syntax:

```
rc = mc.mcFixtureLoadFile(
  number mInst,
  string FileToLoad);
```

Description: Used to load a fixture table into the core.

Parameters:

Parameter	Description
mInst	The controller instance.
FileToLoad	A string buffer specifying the fixture table file.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_FILE_EMPTY	The file did not contain any fixture table data.

Notes:
This function is primarily used by the GUI.

Usage:

```
// Load a fixture table file.
MINSTANCE mInst = 0;
int rc = mcFixtureLoadFile(mInst, "MyFixtureTable.dat");
```

mcFixtureSaveFile

C/C++ Syntax:

```
int mcFixtureSaveFile(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcFixtureSaveFile(
  number mInst),
```

Description: Used to save a previously loaded and modified fixture table file.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_FILE_INVALID	No fixture table file was previously loaded.

Notes:
This function is primarily used by the GUI.

Usage:

```
// Save the fixture table file.
MINSTANCE mInst = 0;
int rc = mcFixtureSaveFile(mInst);
```

mcGuiGetWindowHandle

C/C++ Syntax:

```
int mcGuiGetWindowHandle(
  MINSTANCE mInst,
  void **handle);
```

LUA Syntax:

```
N/A
```

Description: Used by the core or plugins to retriev the GUI's main window handle.

Parameters:

Parameter	Description
mInst	The controller instance.
handle	The address of a void pointer to receive the GUI's main window handle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
If the returned handle is NULL, then no handle was set by the GUI.

Usage:

```
//
MINSTANCE mInst = 0;
void *guiHandle;
rc = mcGuiGetWindowHandle(mInst, &guiHandle;);
```

mcGuiSetCallback

C/C++ Syntax:

```
int mcGuiSetCallback(
 MINSTANCE mInst,
 void *fp);
```

LUA Syntax:

```
N/A
```

Description: Sets a call back function that is used to pass messages from the core to the GUI.

Parameters:

Parameter	Description
mInst	The controller instance.
fp	Function pointer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
If this call back is not set, then the GUI has no means of receiving event based messages.

Usage:

```
/* In the GUI code */
MCP_API int MCP_APIENTRY mcGUIMsg(MINSTANCE mInst, long msg, long wparam, long lp
{
 // Process messages...
 return(MERROR_NOERROR);
}

MINSTANCE mInst = 0;
mcGuiSetCallback(mInst, &mcGUIMsg;);
```

mcGuiSetFocus

C/C++ Syntax:

```
int mcGuiSetFocus(
  MINSTANCE mInst,
  BOOL focus);
```

LUA Syntax:

```
N/A
```

Description: Causes the GUI to gain or lose focus.

Parameters:

Parameter	Description
mInst	The controller instance.
focus	A BOOL specifying the desired GUI focus state. (TRUE/FALSE)

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
The GUI will receive a MSG_GUI_FOCUS message. It is up to the GUI to implement this functionality.

Usage:

```
// Set focus to the GUI's main window
MINSTANCE mInst = 0;
int rc = mcGuiSetFocus(mInst, TRUE);
```

mcGuiSetWindowHandle

C/C++ Syntax:

```
int mcGuiSetWindowHandle(
  MINSTANCE mInst,
  void *handle);
```

LUA Syntax:

```
N/A
```

Description: Informs the core of the GUI's main window handle.

Parameters:

Parameter	Description
mInst	The controller instance.
handle	The GUI's main window handle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
It is up to the GUI to implement this functionality. The handle should be the HWND handle on the Windows platform and the wxWindow handle on all other platforms.

Usage:

```
// Set the GUI main window handle.
MINSTANCE mInst = 0;
 mcGuiSetWindowHandle(0, this);
```

mcToolPathCreate

C/C++ Syntax:

```
int mcToolPathCreate(
  MINSTANCE mInst,
  void *window);
```

LUA Syntax:

```
N/A
```

Description: Create an OpenGL window to display the tool path.

Parameters:

Parameter	Description
mInst	The controller instance.
window	The parent window handle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
This API function is specific to GUI programming.
Passing the window handle is all that is needed to draw the toolpath.
For the Windows Platform, **window** should be the HWND of the parent window. For Linux and Mac, the wxWidgets wxWindow handle must be used.

Usage:

```
wxPanel* m_tpTP;

MINSTANCE mInst = 0;
m_tpTP = new wxPanel(itemPanel129, ID_TOOLPATH_PANEL, wxDefaultPosition, wxDefau
rc = m_tpTP->Show();
// Pass the handle of the tool path.
mcToolPathCreate(mInst, m_tpTP->GetHWND());
```

mcToolPathDelete

C/C++ Syntax:

```
int mcToolPathDelete(
  MINSTANCE mInst,
  void *parent);
```

LUA Syntax:

```
N/A
```

Description:

Parameters:

Parameter	Description
mInst	The controller instance.
parent	The handle of the tool paths parent window.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

This API function is specific to GUI programming.
For the Windows Platform, **window** should be the HWND of the parent window. For Linux and Mac, the wxWidgets wxWindow handle must be used.

Usage:

```
// Delete an existing tool path.
MINSTANCE mInst = 0;
HWND parent; // A previously valid window handle.
int rc = mcToolPathDelete(mInst, parent);
```

mcToolPathGenerate

C/C++ Syntax:

```
int mcToolPathGenerate(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcToolPathGenerate(
  number mInst)
```

Description: Generates the toolpath to refresh any changes that may have been done (like cutter comp or fixure offset changes).

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Generation of the tool path should only be done with the Gocd file is not running.

Usage:

```
MINSTANCE mInst = 0;
// Regenerate the toolpaths for controller instance 0
int rc = mcToolPathGenerate(mInst);
```

mcToolPathGenerateAbort

C/C++ Syntax:

```
int mcToolPathGenerateAbort(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcToolPathGenerateAbort(
  number mInst);
```

Description: Abort tool path generation.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Used if there is an endless loop in the Gcode or if a file regen needs to be stopped.

Usage:

```
MINSTANCE mInst = 0;
// Abort the regeneration of the toolpath for controller 0.
int rc = mcToolPathGenerateAbort(mInst);
```

mcToolPathGeneratedPercent

C/C++ Syntax:

```
int mcToolPathGeneratedPercent(
  MINSTANCE mInst,
  double *percent);
```

LUA Syntax:

```
percent, rc = mc.mcToolPathGeneratedPercent(
  number mInst)
```

Description: Retrieve the tool path generation percentage completed.

Parameters:

Parameter	Description
mInst	The controller instance.
percent	The address of a double to receive the tool path completion percentage. (0 to 100)

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
This function is to be used to show the path generation progress in a progress dialog when a regen has been initiated.

Usage:

```
MINSTANCE mInst = 0;
double pdone = 0;

while (pdone != 100) {
 // Get the percent of the file that is loaded.
 mcToolPathGeneratedPercent(mInst, &pdone;);
 // Post the pdone to some dialog.
 Sleep(250);
}
```

mcToolPathGetAAxisPosition

C/C++ Syntax:

```
mcToolPathGetAAxisPosition(MINSTANCE mInst, double *xPos, double *yPos, double *
```

LUA Syntax:

```
mcToolPathGetAAxisPosition(MINSTANCE mInst, double *xPos, double *yPos, double *
```

Description:

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

Usage:

```
//
MINSTANCE mInst = 0;
mcToolPathGetAAxisPosition(MINSTANCE mInst, double *xPos, double *yPos, double *
```

mcToolPathGetARotationAxis

C/C++ Syntax:

```
mcToolPathGetARotationAxis(MINSTANCE mInst, int *axis);
```

LUA Syntax:

```
mcToolPathGetARotationAxis(MINSTANCE mInst, int *axis);
```

Description:

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

Usage:

```
//
MINSTANCE mInst = 0;
mcToolPathGetARotationAxis(MINSTANCE mInst, int *axis);
```

mcToolPathGetAxisColor

C/C++ Syntax:

```
int mcToolPathGetAxisColor(
  MINSTANCE mInst,
  unsigned long *axiscolor,
  unsigned long *limitcolor);
```

LUA Syntax:

```
axiscolor, limitcolor, rc = mc.mcToolPathGetAxisColor(
  number mInst)
```

Description: Retrieve the current axis and limit tool path colors.

Parameters:

Parameter	Description
mInst	The controller instance.
axiscolor	The address of an unsigned long to receive the axis color.
limitcolor	The address of an unsigned long to receive the sof limit color.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
The colors are in RGB format converted into unsigned longs.

Usage:

```
// Get the axis and limit colors.
MINSTANCE mInst = 0;
int rc = mcToolPathGetAxisColor(MINSTANCE mInst, unsigned long *axiscolor, unsig
```

mcToolPathGetBackColor

C/C++ Syntax:

```
int mcToolPathGetBackColor(
  MINSTANCE mInst,
  unsigned long *topcolor,
  unsigned long *botcolor);
```

LUA Syntax:

```
topcolor, botcolor, rc = mc.mcToolPathGetBackColor(
  number mInst)
```

Description: Get the background top and bottom colors.

Parameters:

Parameter	Description
mInst	The controller instance.
topcolor	The address of an unsigned long that receives the top color of the background.
botcolor	The address of an unsigned long that receives the bottom color of the background.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
The colors are in RGB format converted into unsigned longs.

Usage:

```
MINSTANCE mInst = 0;
unsigned long topcol, botcol;

// Get the tool background top and bottom colors
int rc = mcToolPathGetBackColor(m_inst, &topcol;, &botcol;);
```

mcToolPathGetDrawLimits

C/C++ Syntax:

```
int mcToolPathGetDrawLimits(
  MINSTANCE mInst,
  BOOL *drawlimits);
```

LUA Syntax:

```
drawlimits, rc = mcToolPathGetDrawLimits(
  number mInst)
```

Description: Retrieve the status of the soft limit display.

Parameters:

Parameter	Description
mInst	The controller instance.
drawlimits	The address of a BOOL to receives the state of soft limit display.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
MINSTANCE mInst = 0;
BOOL drawlimits = FALSE;

// See if the soft limits bounding box is beeing shown in the toolpath.
int rc = mcToolPathGetDrawLimits(mInst, &drawlimits;);
```

mcToolPathGetExecution

C/C++ Syntax:

```
int mcToolPathGetExecution(
  MINSTANCE mInst,
  unsigned long exNum,
  void **data,
  unsigned long *len);
```

LUA Syntax:

```
N/A
```

Description:

Parameters:

Parameter	Description
mInst	The controller instance.
exNum	An unsigned long specifying the execution number.
data	The address of a void pointer to receive the move execution info structure.
len	The address of an unsigned long the receive the length of the move execution infomation.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

This structure is not in the MachAPI.h header file. And it is subject to change without notice. Any new items will be added onto the end of the structure thus it is important to use this API call correctly and not rely on the length of the structure given below.

```
struct Executions
{
 bool linear : 1; //Is this a linear move?
 bool rapid : 1;  //Is it a Rapid move?
 bool inc : 1;  //Is this an inc move?
 bool comp : 1;
```

```
 bool rotation : 1;      //arc direction
 bool UsedAxis0 : 1;
 bool UsedAxis1 : 1;
 bool UsedAxis2 : 1;
 bool UsedAxis3 : 1;
 bool UsedAxis4 : 1;
 bool UsedAxis5 : 1;
 bool UsedAxis6 : 1;
 bool UsedAxis7 : 1;
 bool UsedAxis8 : 1;
 bool UsedAxis9 : 1;
 bool UsedAxis10 : 1;
 float end[6];
 float center[3];
 float normal[3];
 unsigned int LineNumber; // Line number in the file...
 unsigned int ExecutionID; // Number of the move from the Gcode interperter
 float FeedRate;   // Feedrate of the move
 unsigned char ToolNumber; // Tool number of the move used to show offsets..
 unsigned char Fixture;  // The fixture.
};
```

Usage:

```
//
MINSTANCE mInst = 0;
unsigned long exNum = 0;
unsigned long len;
void *data;
int rc;
struct Executions *ex;
// Get the length of the move execution structure first.
while (mcToolPathGetExecution(mInst, exNum, NULL, &len;) == MERROR_NOERROR) {
 // Allocate some mem to receive the data.
 data = malloc(len);
 if (data != NULL) {
  // Get the data.
  if (mcToolPathGetExecution(mInst, exNum, &data;, &len;) == MERROR_NOERROR) {
   // cast the void pointer to the current structure.
   ex = (struct Executions *)data;
   ...
   free(data);
  }
 }
}
```

mcToolPathGetFollowMode

C/C++ Syntax:

```
int mcToolPathGetFollowMode(
  MINSTANCE mInst,
  BOOL *enabled);
```

LUA Syntax:

```
enabled, rc = mc.mcToolPathGetFollowMode(
  number mInst)
```

Description: Retrieve the status of the tool path jog follow mode.

Parameters:

Parameter	Description
mInst	The controller instance.
enabled	The address of a BOOL to receive the jog follow mode state.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// See if jog follow is enabled.
MINSTANCE mInst = 0;
BOOL enabled = FALSE;
int rc = mcToolPathGetFollowMode(mInst, &enabled;);
```

mcToolPathGetGenerating

C/C++ Syntax:

```
int mcToolPathGetGenerating(
  MINSTANCE mInst,
  int *pathGenerating);
```

LUA Syntax:

```
pathGenerating, rc = mc.mcToolPathGetGenerating(
  number mInst)
```

Description: Retrieve the state of the tool path generation.

Parameters:

Parameter	Description
mInst	The controller instance.
pathGenerating	The address A pointer to an int to receive the path generation state (MC_ON, MC_OFF).

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**pathGenerating** is NULL.

Notes:
This function is used after a file load, or regen to keep the user from using any controles as the machine is loading.

Usage:

```
MINSTANCE mInst = 0;
BOOL IsGen = FALSE;
// See if the tool path is generating.
int rc = mcGetPathGenerating(mInst, &IsGen;);
```

mcToolPathGetLeftMouseDn

C/C++ Syntax:

```
int mcToolPathGetLeftMouseDn(
  MINSTANCE mInst,
  double *x,
  double *y,
  double *z);
```

LUA Syntax:

```
x, y, z, rc = mc.mcToolPathGetLeftMouseDn(
  number mInst)
```

Description: Retrieve the xyz coordinates of the mouse in the tool path when the left button goes down.

Parameters:

Parameter	Description
mInst	The controller instance.
x	The address to a double to revceive the x coordinate.
y	The address to a double to revceive the y coordinate.
z	The address to a double to revceive the z coordinate.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Get the mouse coordinates in the tool path.
MINSTANCE mInst = 0;
double x, y, z;
int rc = mcToolPathGetLeftMouseDn(MINSTANCE mInst, double *x, double *y, double
```

mcToolPathGetLeftMouseUp

C/C++ Syntax:

```
int mcToolPathGetLeftMouseUp(
  MINSTANCE mInst,
  double *x,
  double *y,
  double *z);
```

LUA Syntax:

```
x, y, z, rc = mc.mcToolPathGetLeftMouseUp(
  number mInst)
```

Description: Retrieve the xyz coordinates of the mouse in the tool path when the left button comes up.

Parameters:

Parameter	Description
mInst	The controller instance.
x	The address to a double to revceive the x coordinate.
y	The address to a double to revceive the y coordinate.
z	The address to a double to revceive the z coordinate.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Get the mouse coordinates in the tool path.
MINSTANCE mInst = 0;
double x, y, z;
int rc = mcToolPathGetLeftMouseUp(MINSTANCE mInst, double *x, double *y, double
```

mcToolPathGetPathColor

C/C++ Syntax:

```
int mcToolPathGetPathColor(
  MINSTANCE mInst,
  unsigned long *rapidcolor,
  unsigned long *linecolor,
  unsigned long *arccolor,
  unsigned long *highlightcolor);
```

LUA Syntax:

```
rapidcolor, linecolor, arccolor, highlightcolor, rc = mcToolPathGetPathColor(
  number mInst)
```

Description: Retrieve the tool path display colors.

Parameters:

Parameter	Description
mInst	The controller instance.
rapidcolor	The address of an unsigned long that receives the rapid (G00) color of the toolpath.
linecolor	The address of an unsigned long that receives the line (G01) color of the toolpath.
arccolor	The address of an unsigned long that receives the arc (G02/G03) color of the toolpath.
highlightcolor	The address of an unsigned long that receives the past moves color of the toolpath.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
The colors are in RGB format converted into unsigned longs.

Usage:

```
MINSTANCE mInst = 0;
unsigned long rapcol, lincol, arccol, highcol;
// Get the tool path colors
int rc = mcToolPathGetPathColor(m_inst, &rapcol;, &lincol;, &arccol;, &highcol;)
```

mcToolPathIsSignalMouseClicks

C/C++ Syntax:

```
int mcToolPathIsSignalMouseClicks(
  MINSTANCE mInst,
  BOOL *enabled);
```

LUA Syntax:

```
enabled, rc = mcToolPathIsSignalMouseClicks(
  number mInst)
```

Description: Determine if tool path mouse click events are signaled.

Parameters:

Parameter	Description
mInst	The controller instance.
enabled	The address of a BOOL to receive the state of the tool path mouse clicks.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

Usage:

```
// See if we are signaling mouse click events in the tool path.
MINSTANCE mInst = 0;
BOOL enabled = FALSE;
int rc = mcToolPathIsSignalMouseClicks(mInst, &enabled;);
```

mcToolPathSetAAxisPosition

C/C++ Syntax:

```
mcToolPathSetAAxisPosition(MINSTANCE mInst, double xPos, double yPos, double zPos
```

LUA Syntax:

```
mcToolPathSetAAxisPosition(MINSTANCE mInst, double xPos, double yPos, double zPos
```

Description:

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

Usage:

```
//
MINSTANCE mInst = 0;
mcToolPathSetAAxisPosition(MINSTANCE mInst, double xPos, double yPos, double zPos
```

mcToolPathSetARotationAxis

C/C++ Syntax:

```
mcToolPathSetARotationAxis(MINSTANCE mInst, int axis);
```

LUA Syntax:

```
mcToolPathSetARotationAxis(MINSTANCE mInst, int axis);
```

Description:

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

Usage:

```
//
MINSTANCE mInst = 0;
mcToolPathSetARotationAxis(MINSTANCE mInst, int axis);
```

mcToolPathSetAxisColor

C/C++ Syntax:

```
int mcToolPathSetAxisColor(
  MINSTANCE mInst,
  unsigned long axiscolor,
  unsigned long limitcolor);
```

LUA Syntax:

```
rc = mc.mcToolPathSetAxisColor(
  number mInst,
  number axiscolor,
  number limitcolor)
```

Description: Set the tool path axis and limit colors.

Parameters:

Parameter	Description
mInst	The controller instance.
axiscolor	The color of the top of the axis lines in the toolpath.
limitcolor	The color of the softlimits color bounding box in the toolpath.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
The colors are in RGB format converted into unsigned long.

Usage:

```
MINSTANCE mInst = 0;
unsigned long axiscolor;
unsigned long limitcolor;

//red
axiscolor =  (255<<0);//red
axiscolor += (0<<8);//green
```

```
axiscolor += (0<<16);//blue

//Yellow
limitcolor =  (255<<0);//red
limitcolor += (255<<8);//green
limitcolor += (128<<16);//blue

// Set the axis and limit colors
int rc = mcToolPathSetAxisColor(mInst, axiscolor, limitcolor);
```

mcToolPathSetBackColor

C/C++ Syntax:

```
int mcToolPathSetBackColor(
  MINSTANCE mInst,
  unsigned long topcolor,
  unsigned long botcolor);
```

LUA Syntax:

```
rc = mc.mcToolPathSetBackColor(
  number mInst,
  number topcolor,
  number botcolor)
```

Description: Set the tool path background color.

Parameters:

Parameter	Description
mInst	The controller instance.
topcolor	The color of the top of the background.
botcolor	The color of the bottom of the background.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Used to set the top and botom background color of toolpath. The colors are in RGB format converted into unsigned long.

Usage:

```
MINSTANCE mInst = 0;
unsigned long topcolor;
unsigned long botcolor

//Dark Blue Top
topcolor = (28<<0);
topcolor += (28<<8);
```

```
topcolor += (213<<16);
//Light Blue Bottom
botcolor = (164<<0);
botcolor += (156<<8);
botcolor += (228<<16);

// Set the background color
int rc = mcToolPathSetBackColor(mInst, topcolor, botcolor);
```

mcToolPathSetDrawLimits

C/C++ Syntax:

```
int mcToolPathSetDrawLimits(
  MINSTANCE mInst,
  BOOL drawlimits);
```

LUA Syntax:

```
rc = mc.mcToolPathSetDrawLimits(
  number mInst,
  number drawlimits)
```

Description: Enable or disable the drawing of soft limits on the tool paths.

Parameters: (drawlimits, A BOOL specifying whether soft limits are drawn on the tool path.

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Turn on soft limits in the tool path.
MINSTANCE mInst = 0;
int rc = mcToolPathSetDrawLimits(mInst, TRUE);
```

mcToolPathSetFollowMode

C/C++ Syntax:

```
int mcToolPathSetFollowMode(
  MINSTANCE mInst,
  BOOL enabled);
```

LUA Syntax:

```
rc = mc.mcToolPathSetFollowMode(
  number mInst,
  number enabled)
```

Description: Enable or disable tool path jog follow mode.

Parameters:

Parameter	Description
mInst	The controller instance.
enabled	A BOOL specifying the state of the tool path jog follow mode.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Turn on tool path jog following.
MINSTANCE mInst = 0;
int rc = mcToolPathSetFollowMode(mInst, TRUE);
```

mcToolPathSetPathColor

C/C++ Syntax:

```
int mcToolPathSetPathColor(
  MINSTANCE mInst,
  unsigned long rapidcolor,
  unsigned long linecolor,
  unsigned long arccolor,
  unsigned long highlightcolor);
```

LUA Syntax:

```
rc = mc.mcToolPathSetPathColor(
  number mInst,
  number rapidcolor,
  number linecolor,
  number arccolor,
  number highlightcolor)
```

Description: Set the tool path line display colors.

Parameters:

Parameter	Description
mInst	The controller instance.
rapidcolor	The color of the rapid (G00) moves in the toolpath.
linecolor	The color of the linear feed (G01) moves in the toolpath.
arccolor	The color of the arc (G03/G02) moves in the toolpath.
highlightcolor	The color of the moves that have been done in the toolpath.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
The colors are in RGB format converted into unsigned longs.

Usage:

```
// Set the tool path colors
MINSTANCE mInst = 0;
unsigned long rapidcolor;
unsigned long linecolor;
unsigned long arccolor;
unsigned long highlightcolor;

rapidcolor =  (254<<0);//Red
rapidcolor += (0<<8);//Green
rapidcolor += (0<<16);//Blue

linecolor =  (0<<0);//Red
linecolor += (254<<8);//Green
linecolor += (0<<16);//Blue

arccolor =  (0<<0);//Red
arccolor += (254<<8);//Green
arccolor += (0<<16);//Blue

highlightcolor =  (254<<0);//Red
highlightcolor += (254<<8);//Green
highlightcolor += (254<<16);//Blue

int rc = mcToolPathSetPathColor(mInst, rapidcolor, linecolor, arccolor, highlight
```

mcToolPathSetSignalMouseClicks

C/C++ Syntax:

```
int mcToolPathSetSignalMouseClicks(
  MINSTANCE mInst,
  BOOL enabled);
```

LUA Syntax:

```
rc = mc.mcToolPathSetSignalMouseClicks(
  number mInst,
  number enabled)
```

Description: Enable or Disable the tool path mouse click events.

Parameters:

Parameter	Description
mInst	The controller instance.
enabled	A BOOL specifying the state of the tool path mouse click event signaling.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

Usage:

```
// Turn on mouse click events in the tool path.
MINSTANCE mInst = 0;
int rc = mcToolPathSetSignalMouseClicks(mInst, TRUE);
```

mcToolPathSetView

C/C++ Syntax:

```
int mcToolPathSetView(
  MINSTANCE mInst,
  void *parent,
  int view);
```

LUA Syntax:

```
N/A
```

Description: Set the tool path view orientation.

Parameters:

Parameter	Description
mInst	The controller instance.
parent	The tool path parent window handle.
view	An integer specifying the tool path view orientation. (MC_TPVIEW_TOP, MC_TPVIEW_BOTTOM, MC_TPVIEW_LEFT, MC_TPVIEW_RIGHT, or MC_TPVIEW_ISO)

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
This API function is specific to GUI programming.
For the Windows Platform, **window** should be the HWND of the parent window. For Linux and Mac, the wxWidgets wxWindow handle must be used.

Usage:

```
// Set the tool path orientation to the ISO view.
MINSTANCE mInst = 0;
HWND parent;  // A valid window handle.
int rc = mcToolPathSetView(mInst, parent, MCTPVIEW_ISO);
```

mcToolPathSetZoom

C/C++ Syntax:

```
int mcToolPathSetZoom(
  MINSTANCE mInst,
  void *parent,
  double zoom);
```

LUA Syntax:

```
N/A
```

Description: Set the zoom of the given tool path

Parameters:

Parameter	Description
mInst	The controller instance.
parent	The tool path parent window handle.
zoom	A double specifying the zoom percentage as a decimal.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
This API function is specific to GUI programming.
For the Windows Platform, **window** should be the HWND of the parent window. For Linux and Mac, the wxWidgets wxWindow handle must be used.
zoom should be a decimal percentage. e.g. 1.5 == 150%.

Usage:

```
// Set the toop path zoom percentage to 150%
MINSTANCE mInst = 0;
HWND parent;  // A valid window handle.
int rc = mcToolPathSetZoom(mInst, parent, 1.5);
```

mcToolPathUpdate

C/C++ Syntax:

```
int mcToolPathUpdate(
  MINSTANCE mInst,
  void *parent);
```

LUA Syntax:

```
N/A
```

Description: Redraw/update the OpenGL tool path.

Parameters:

Parameter	Description
mInst	The controller instance.
parent	The tool path parent window handle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
No Error.	
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
The mInst parameter was out of range.	

Notes:
Updates need to be run at every .1 sec or faseter to have a smooth tool path.
This API function is specific to GUI programming.
For the Windows Platform, **window** should be the HWND of the parent window. For Linux and Mac, the wxWidgets wxWindow handle must be used.

Usage:

```
MINSTANCE mInst = 0;
HWND parent;  // A valid window handle.
int mcToolPathUpdate(mInst, parent);
```

Profile (INI) Settings

mcProfileDeleteKey

C/C++ Syntax:

```
int mcProfileDeleteKey(
  MINSTANCE mInst,
  const char *section,
  const char *key);
```

LUA Syntax:

```
rc = mc.mcProfileDeleteKey(
  number mInst,
  string section,
  string key)
```

Description: Delete key from the profile's INI file.

Parameters:

Parameter	Description
mInst	The controller instance.
section	A pointer to a char buffer with section string.
key	A pointer to a char buffer with key string.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	One or more of the parameters had a NULL value.

Notes:
None.

Usage:

```
MINSTANCE mInst = 0;
mcProfileDeleteKey(mInst , "P_Port", "Frequency");
```

mcProfileDeleteSection

C/C++ Syntax:

```
int mcProfileDeleteSection(
  MINSTANCE mInst,
  const char *section);
```

LUA Syntax:

```
rc = mc.mcProfileDeleteSection(
  number mInst,
  string section)
```

Description: Delete the specified section from the profile's INI file.

Parameters:

Parameter	Description
mInst	The controller instance.
section	A string buffer buffer specifying the section.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	One or more of the parameters had a NULL value.

Notes:
None.

Usage:

```
MINSTANCE mInst = 0;
mcProfileDeleteKey(mInst , "P_Port");
```

mcProfileExists

C/C++ Syntax:

```
int mcProfileExists(
  MINSTANCE mInst,
  const char *section,
  const char *key);
```

LUA Syntax:

```
rc = mc.mcProfileExists(
  number mInst,
  string section,
  string key)
```

Description: Determine if the section or section and key are in the profiles INI file.

Parameters:

Parameter	Description
mInst	The controller instance.
section	A string buffer specifying the section.
key	A string buffer specifying the key.

Returns:

Return Code	Description
MC_TRUE	The item pecified by **section** and **key** was found in INI file.
MC_FALSE	The item pecified by **section** and **key** was not found in INI file.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**section** cannot be NULL.

Notes:
If you wish to check only for the existence of a section, use NULL for the value of **key**.

Usage:

```
MINSTANCE mInst = 0;
mcProfileWriteInt(mInst, "P_Port", "Frequency");
```

mcProfileGetDouble

C/C++ Syntax:

```
mcProfileGetDouble(
  MINSTANCE mInst,
  const char *section,
  const char *key,
  double *retval,
  double defval);
```

LUA Syntax:

```
retval, rc = mc.mcProfileGetDouble(
  number mInst,
  string section,
  string key,
  number defval)
```

Description: Read a double value from the profile's INI file.

Parameters:

Parameter	Description
mInst	The controller instance.
section	A string buffer specifying the section.
key	A string buffer specifying the key.
retval	The address of a double to receive the value from the section and key location.
defval	The default value if the item is not found.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	One or more of the parameters had a NULL value.

Notes:
Reading the profile settings can be done at any time. If **section** and **key** are not in the file, they are created and assigned the default value.

Usage:

```
double rval=0;
MINSTANCE mInst = 0;
mcProfileGetDouble(mInst , "P_Port", "Frequency", &rval;, 25.34);
```

mcProfileGetInt

C/C++ Syntax:

```
int mcProfileGetInt(
  MINSTANCE mInst,
  const char *section,
  const char *key,
  long *retval,
  long defval);
```

LUA Syntax:

```
retval, rc = mc.mcProfileGetInt(
  number mInst,
  string section,
  string key,
  number defval)
```

Description: Read a long value from the profile's INI file.

Parameters:

Parameter	Description
mInst	The controller instance.
section	A string buffer specifying the section.
key	A string buffer specifying the key.
retval	The address of a long to receive the value from the section and key location.
defval	The default value if the item is not found.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
No Error.	
MERROR_INVALID_ARG	One or more of the parameters had a NULL value.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
The mInst parameter was out of	

range.

Notes:
Reading the profile settings can be done at any time. If **section** and **key** are not in the file, they are created and assigned the default value.

Usage:

```
long rval=0;
int mInst=0;
mcProfileGetInt(mInst , "P_Port", "Frequency", &rval;, 25000);
```

mcProfileGetName

C/C++ Syntax:

```
int mcProfileGetName(
  MINSTANCE mInst,
  char *buff,
  size_t bufsize);
```

LUA Syntax:

```
name, rc = mc.mcProfileGetName(
  number mInst)
```

Description: Retrieve the current profile name.

Parameters:

Parameter	Description
mInst	The controller instance.
buff	A string buffer to receive the profile's name.
buffsize	The length of the buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	One or more of the parameters had a NULL value.

Notes:
None

Usage:

```
MINSTANCE mInst = 0;
char buff[80];
memset(buff, 0, 80);
mcProfileGetName(mInst, buff, 80);
```

mcProfileGetString

C/C++ Syntax:

```
int mcProfileGetString(
  MINSTANCE mInst,
  const char *section,
  const char *key,
  char *buff,
  long buffsize,
  const char *defval);
```

LUA Syntax:

```
buff, rc = mc.mcProfileGetString(
  MINSTANCE mInst,
  string section,
  string key,
  string defval);
```

Description: Read a string value from the profile's INI file.

Parameters:

Parameter	Description
mInst	The controller instance.
section	A string buffer specifying the section.
key	A string buffer specifying the key.
buff	A string buffer to receive the string from the section and key location.
buffsize	The length of the buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	One or more of the parameters had a NULL value.

Notes:
Reading the profile settings can be done at any time. If **section** and **key** are not in the file, they are

created and assigned the default value.

Usage:

```
MINSTANCE m_inst = 0;
char *key = "BufferedTime";
char buff[80];
memset(buff, 0, 80);
mcProfileGetString(mInst , "SomeSection", key, buff, 80, "0.100");
```

mcProfileSave

C/C++ Syntax:

```
int mcProfileSave(
  INSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcProfileSave(
  number mInst)
```

Description: Save the profile's settings to the INI file.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Saving the profile will write the settings to the profile's INI file. This will ensure that a power failue will not result in lost settings. The settings that are saved are the core settings. Plugin and GUI settings fall under the responsability of the GUI or plugin programmer.

Usage:

```
MINSTANCE mInst = 0;
mcProfileSave(mInst); // Flush the settings to the INI file.
```

mcProfileWriteDouble

C/C++ Syntax:

```
int mcProfileWriteDouble(
  MINSTANCE mInst,
  const char *section,
  const char *key,
  double val);
```

LUA Syntax:

```
rc = mc.mcProfileWriteDouble(
  number mInst,
  string section,
  string key,
  double val)
```

Description: Write a double value to the profile's INI file.

Parameters:

Parameter	Description
mInst	The controller instance.
section	A string buffer specifying the section.
key	A string buffer specifying the key.
val	A double specifying the value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	One or more of the parameters had a NULL value.

Notes:
Writing to the profile settings can be done at any time.

Usage:

```
MINSTANCE mInst=0;
```

```
mcProfileWriteDouble(m_cid , "P_Port", "Frequency", 45000);
```

mcProfileWriteInt

C/C++ Syntax:

```
int mcProfileWriteInt(
  MINSTANCE mInst,
  const char *section,
  const char *key,
  long val);
```

LUA Syntax:

```
int mcProfileWriteInt(
  number mInst,
  string section,
  string key,
  number val)
```

Description: Write a long value to the profile's INI file.

Parameters:

Parameter	Description
mInst	The controller instance.
section	A string buffer specifying the section.
key	A string buffer specifying the key.
val	A long specifying the value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	One or more of the parameters had a NULL value.

Notes:
Writing to the profile settings can be done at any time.

Usage:

```
MINSTANCE mInst = 0;
```

```
mcProfileWriteInt(m_cid , "P_Port", "Frequency", 45000);
```

mcProfileWriteString

C/C++ Syntax:

```
int mcProfileWriteString(
  MINSTANCE mInst,
  const char *section,
  const char *key,
  const char *val);
```

LUA Syntax:

```
rc = mc.mcProfileWriteString(
  number mInst,
  string section,
  string key,
  string val)
```

Description: Write a string value to the profile's INI file.

Parameters:

Parameter	Description
mInst	The controller instance.
section	A string buffer specifying the section.
key	A string buffer specifying the key.
val	A string buffer specifying the value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	One or more of the parameters had a NULL value.

Notes:
Writing to the profile settings can be done at any time.

Usage:

```
MINSTANCE m_inst = 0;
```

```
char *key = "BufferedTime"
double BuffTime = .250;
char val[80];
sprintf(val, "%.4f", BuffTime);
mcProfileWriteString(mInst, "Darwin", key, val);
```

General

- mcCntlPluginConfig
- mcCntlPluginDiag
- mcCntlProbeFileClose
- mcCntlProbeFileOpen
- mcCntlSetCoolantDelay
- mcCntlSetDiaMode
- mcCntlSetMistDelay
- mcCntlSetMode
- mcCntlSetPoundVar
- mcCntlSetStats
- mcCntlSetValue
- mcFileHoldAquire
- mcFileHoldReason
- mcFileHoldRelease

mcCntlConfigStart

C/C++ Syntax:

```
int mcCntlConfigStart(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcCntlConfigStart(
  number mInst);
```

Description: Put the control into the configure state.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_STATE	The control is in a state from which it cannot transition to the configure state.

Notes:
Upon entering the configure state, MSG_CONFIG_START is sent to the GUI and plugins.

Usage:

```
// Enter the configure state
MINSTANCE mInst = 0;
if (mcCntlConfigStart(mInst) == MERRROR_NOERROR) {
 // Successfully entered the configure state!
}
```

mcCntlConfigStop

C/C++ Syntax:

```
int mcCntlConfigStop(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcCntlConfigStop(
  number mInst);
```

Description: Leave the configure state.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_STATE	The control was not in the configure state.

Notes:
Upon leaving the congifure state, MSG_CONFIG_END is sent to the GUI and plugins.

Usage:

```
// Leave the configure state.
MINSTANCE mInst = 0;
int rc = mcCntlConfigStop(mInst);
if (rc == MERROR_NOERROR) {
 // Successfully exited the configure state.
}
```

mcCntlEnable

C/C++ Syntax:

```
int mcCntlEnable(
 MINSTANCE mInst,
 BOOL state);
```

LUA Syntax:

```
rc = mc.mcCntlEnable(
 number mInst,
 number state)
```

Description: Enable or disable the control.

Parameters:

Parameter	Description
mInst	The controller instance.
state	A BOOL specifying the desired control state. TRUE for enabled

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_NOW	The operation could not be completed at this time.
The ISIG_EMERGENCY signal is asserted.	

Notes:
If ISIG_EMERGENCY is asserted, the e-stop condition must be cleared.

Usage:

```
// Enable the control.
MINSTANCE mInst = 0;
int rc = mcCntlEnable(mInst, TRUE);
```

mcCntlGcodeExecute

C/C++ Syntax:

```
int mcCntlGcodeExecute(
  MINSTANCE mInst,
  const char *commands);
```

LUA Syntax:

```
rc = mc.mcCntlGcodeExecute(
  number mInst,
  string commands)
```

Description: Execute G code as a unit of work.

Parameters:

Parameter	Description
mInst	The controller instance.
commands	A string buffer containing G code.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_NOW	The operation could not be completed at this time.
MERROR_NOT_COMPILED	The macro scripts could not be compiled.

Notes:

If the control is in the idle state, this function will execute the G code in MDI mode.

It is important to note that this function does not wait for the G code to complete. It is an error to execute G code with this function one line after another. Instead, the lines should be combined into one string separeated with newline characters (\n).

Usage:

```
// Execute spindle stop and rapid to 0, 0.
MINSTANCE mInst = 0;
int rc;
rc = mcCntlGcodeExecute(mInst, "M05\nG00 X0 Y0");
if (rc == MERROR_NOERROR) {
 // The G code was submitted for processing.
// However, the function return immediately and does not wait on the G code to f
}
```

mcCntlGcodeExecuteWait

C/C++ Syntax:

```
int mcCntlGcodeExecuteWait(
  MINSTANCE mInst,
  const char *commands);
```

LUA Syntax:

```
rc = mc.mcCntlGcodeExecute(
  number mInst,
  string commands)
```

Description: Execute G code as a unit of work and wait until it is complete.

Parameters:

Parameter	Description
mInst	The controller instance.
commands	A string buffer containing G code.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_NOW	The operation could not be completed at this time.
MERROR_NOT_COMPILED	The macro scripts could not be compiled.

Notes:

If the control is in the idle state, this function will execute the G code in MDI mode.

Usage:

```
// Execute spindle stop and rapid to 0, 0.
MINSTANCE mInst = 0;
int rc;
rc = mcCntlGcodeExecute(mInst, "M05\nG00 X0 Y0");
if (rc == MERROR_NOERROR) {
```

```
 // The G code was submitted for processing and has completed.
}
```

mcCntlGcodeInterpGetData

C/C++ Syntax:

```
int mcCntlGcodeInterpGetData(
  MINSTANCE mInst,
  interperter_t *data);
```

LUA Syntax:

```
N/A
```

Description: Retrieves the current G code interpreter data.

Parameters:

Parameter	Description
mInst	The controller instance.
data	The address of a Interpreter_info struct.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

```
struct Interperter_info {
 double ModalGroups[MC_MAX_GROUPS];
 double FeedRate;
 double SpindleSpeed;
 int SpindleDirection;
 BOOL Mist;
 BOOL Flood;
 int ToolNumber;
 int HeightRegister;
 int DiaRegister;
};
typedef struct Interperter_info interperter_t;
```

Usage:

```
// Get the interpreter information.
MINSTANCE mInst = 0;
```

```
interperter_t data;
int rc;
rc = mcCntlGcodeInterpGetData(mInst, &data;);
if (rc == MERROR_NOERROR) {
 // Success!
}
```

mcCntlGcodeInterpGetPos

C/C++ Syntax:

```
int mcCntlGcodeInterpGetPos(
  MINSTANCE mInst,
  int axisId,
  double *pos);
```

LUA Syntax:

```
pos, rc = mc.mcCntlGcodeInterpGetPos(
  number mInst,
  number axisId)
```

Description: Retrieves the current G code interpreter position for the given axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	An integer specifying the axis.
pos	An address of a double to receive the axis' position.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
It is important to not that the look ahead buffer will affect what value is returned and it may not be the current position.

Usage:

```
// Get the current buffered X axis position.
MINSTANCE mInst = 0;
double pos;
int rc;
rc = int mcCntlGcodeInterpGetPos(mInst, X_AXIS, &pos;);
if (rc == MERROR_NOERROR) {
 // Success!
}
```

mcCntlGetBuild

C/C++ Syntax:

```
int mcCntlGetBuild(
  MINSTANCE mInst,
  char *buf,
  size_t bufsize);
```

LUA Syntax:

```
build, rc = mc.mcCntlGetBuild(
  number mInst)
```

Description: Retrieves the core build number.

Parameters:

Parameter	Description
mInst	The controller instance.
buf	A string buffer that receives the build number.
bufsize	The length of the supplied string buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

Usage:

```
// Get the core's build number
MINSTANCE mInst = 0;
char buildBuf[80];
int rc = mcCntlGetBuild(mInst, buildBuf, sizeof(buildBuf));
if (rc == MERROR_NOERROR) {
 printf("The current build is %s.n", buildBuf);
}
```

mcCntlGetComputerID

C/C++ Syntax:

```
int mcCntlGetComputerID(
  MINSTANCE mInst,
  char *buf,
  size_t bufSize);
```

LUA Syntax:

```
buf, rc = mc.mcCntlGetComputerID(
  number mInst)
```

Description: Retrieve the computer ID(s) for the current host PC.

Parameters:

Parameter	Description
mInst	The controller instance.
buf	A string buffer to receive the ID(s).
bufSize	The size of the string buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Depending on the number of Ethernet cards in the PC, this functions may return more than one ID. If more than one ID is returned, they will be delimited by tab characters.

Usage:

```
// Get the ID(s) for this host
MINSTANCE mInst = 0;
int rc = mcCntlGetComputerID(mInst, buf, sizeof(buf));
if (rc == MERROR_NOERROR) {
 // Parse buf to get the ID(s)
}
```

mcCntlGetCoolantDelay

C/C++ Syntax:

```
int mcCntlGetCoolantDelay(
  MINSTANCE mInst,
  double *secs);
```

LUA Syntax:

```
sec, rc = mc.mcCntlGetCoolantDelay(
  number mInst)
```

Description: Get the coolant delay time in seconds.

Parameters:

Parameter	Description
mInst	The controller instance.
secs	an address of a double to receive the coolant delay.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Get the coolant delay.
MINSTANCE mInst = 0;
double secs;
int rc = mcCntlGetCoolantDelay(mInst, &secs;);
if (rc == MERROR_NOERROR) {
 // Success!
}
```

mcCntlGetCwd

C/C++ Syntax:

```
int mcCntlGetCwd(
  MINSTANCE mInst,
  char *buf,
  size_t bufSize);
```

LUA Syntax:

```
buf, rc = mc.mcCntlGetCwd(
  number mInst)
```

Description: Retrieves the core's current working directory.

Parameters:

Parameter	Description
mInst	The controller instance.
buf	A string buffer to receive the ID(s).
bufSize	The size of the string buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

Usage:

```
// Get current working directory.
MINSTANCE mInst = 0;
char curDir[255];
int rc = mcCntlGetCwd(mInst, curDir, sizeof(curDir));
if (rc == MERROR_NOERROR) {
 // Success!
}
```

mcCntlGetDiaMode

C/C++ Syntax:

```
int mcCntlGetDiaMode(
  MINSTANCE mInst,
  BOOL *dia);
```

LUA Syntax:

```
dia, rc = mc.mcCntlGetDiaMode(
  number mInst)
```

Description: Retrieve the lathe diameter mode.

Parameters:

Parameter	Description
mInst	The controller instance.
dia	The address of a BOOL that receives the diameter mode. (TRUE is diameter mode

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Only valid when the control is in lathe (turn) mode.

Usage:

```
// Get the lathe diameter mode.
MINSTANCE mInst = 0;
BOOL Dia;
int rc = mcCntlGetDiaMode(mInst, &Dia;);
if (rc == MERROR_NOERROR) {
 // Success!
}
```

mcCntlGetDistToGo

C/C++ Syntax:

```
int mcCntlGetDistToGo(
 MINSTANCE mInst,
 int axisId,
 double *togo);
```

LUA Syntax:

```
togo, rc = int mcCntlGetDistToGo(
 number mInst,
 number axisId)
```

Description: Returns the distance from the end point of a move in current block.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis from which to to get the distance.
togo	A pointer to a double to receive the distance left in current move.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.
MERROR_NOT_NOW	The operation could not be completed at this time.

Notes:
Only used when running G code.

Usage:

```
int mInst=0;
int axis = Y_AXIS;
```

```
double togo=0;
mcCntlGetDistToGo(mInst, axis, &togo;);
```

mcCntlGetLocalVar

int mcCntlGetLocalVar(MINSTANCE mInst, HMCVARS hVars, int varNumber, doubl *retval);

LUA Syntax:

```
retval, rc = mc.mcCntlGetLocalVar(
  number mInst,
  number hVars,
  number varNumber)
```

Description:

Retrieve the specified local variable.

Parameters:

Parameter	Description
mInst	The controller instance.
hVars	The handle to the loacl variable stack.
varNumber	The index of the local variable flage to retrieve.
retval	An address of a double integer to receive the value of the specified local variable.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**varNumber** is out of range or **retval** is NULL.

Notes:

This is primarily a script function.

Usage:

```
-- Get local variables.
function m700(hVars)
 local inst = mc.mcGetInstance() -- Get the current instance
 local nilPoundVar = mc.mcCntlGetPoundVar(inst,0)
 local message = ""
 if hVars ~= nil then
  local flag, retval, rc
```

```
  flag, rc = mc.mcCntlGetLocalVarFlag(inst, hVars, mc.SV_A)
  if (flag == 1) then
   retval, rc = mc.mcCntlGetLocalVar(inst, hVars, mc.SV_A)
   if rc == mc.MERROR_NOERROR then
    message = message .. "A" .. ":" .. tostring(retval) .. ", "
   end
  end
  flag, rc = mc.mcCntlGetLocalVarFlag(inst, hVars, mc.SV_A)
  if (flag == 1) then
   retval, rc = mc.mcCntlGetLocalVar(inst, hVars, mc.SV_B)
   if rc == mc.MERROR_NOERROR then
    message = message .. "B" .. ":" .. tostring(retval)
   end
  end
  mc.mcCntlSetLastError(inst, message)
 end
end

if (mc.mcInEditor() == 1) then
    m700(nil) -- We can't test this in the editor!
end
```

mcCntlGetLocalVarFlag

C/C++ Syntax:

```
int mcCntlGetLocalVarFlag(
  MINSTANCE mInst,
  HMCVARS hVars,
  int varNumber,
  int *retval);
```

LUA Syntax:

```
retval, rc = mc.mcCntlGetLocalVarFlag(
  number mInst,
  number hVars,
  number varNumber)
```

Description: Retrieve the specified local variable.

Parameters:

Parameter	Description
mInst	The controller instance.
hVars	The handle to the loacl variable stack.
varNumber	The index of the local variable flage to retrieve.
retval	An address of an integer to receive the value of the specified local variable flag.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**varNumber** is out of range or **retval** is NULL.

Notes:
This is primarily a script function.

Usage:

```
-- Get local variables.
```

```
function m700(hVars)
 local inst = mc.mcGetInstance() -- Get the current instance
 local nilPoundVar = mc.mcCntlGetPoundVar(inst,0)
 local message = ""
 if hVars ~= nil then
  local flag, retval, rc
  flag, rc = mc.mcCntlGetLocalVarFlag(inst, hVars, mc.SV_A)
  if (flag == 1) then
   retval, rc = mc.mcCntlGetLocalVar(inst, hVars, mc.SV_A)
   if rc == mc.MERROR_NOERROR then
    message = message .. "A" .. ":" .. tostring(retval) .. ", "
   end
  end
  flag, rc = mc.mcCntlGetLocalVarFlag(inst, hVars, mc.SV_A)
  if (flag == 1) then
   retval, rc = mc.mcCntlGetLocalVar(inst, hVars, mc.SV_B)
   if rc == mc.MERROR_NOERROR then
    message = message .. "B" .. ":" .. tostring(retval)
   end
  end
  mc.mcCntlSetLastError(inst, message)
 end
end

if (mc.mcInEditor() == 1) then
    m700(nil) -- We can't test this in the editor!
end
```

mcCntlGetLogging

C/C++ Syntax:

```
int mcCntlGetLogging(
  MINSTANCE mInst,
  BOOL *enabled);
```

LUA Syntax:

```
enabled, rc = mc.mcCntlGetLogging(
  number mInst)
```

Description: Determine if logging is enabled.

Parameters: (enabled, The address of a BOOL to receive the logging state.

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**enabled** is NULL.

Notes:
None.

Usage:

```
// Check is logging is enabled.
MINSTANCE mInst = 0;
BOOL enabled = FALSE;
int rc = mcCntlGetLogging(mInst, &enabled;);
if (rc == MERROR_NOERROR && enabled == TRUE) {
 // Logging is enabled!
}
```

mcCntlGetMachDir

C/C++ Syntax:

LUA Syntax:

Description:

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

Usage:

```
//
MINSTANCE mInst = 0;
```

mcCntlGetMistDelay

C/C++ Syntax:

```
int mcCntlGetMistDelay(
  MINSTANCE mInst,
  double *secs);
```

LUA Syntax:

```
secs, rc = mcCntlGetMistDelay(
  number mInst)
```

Description: Get the mist delay time in seconds.

Parameters:

Parameter	Description
mInst	The controller instance.
secs	an address of a double to receive the mist delay.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Get the mist delay.
MINSTANCE mInst = 0;
double secs;
int rc = mcCntlGetMistDelay(mInst, &secs;);
if (rc == MERROR_NOERROR) {
 // Success!
}
```

mcCntlGetModalGroup

C/C++ Syntax:

```
int mcCntlGetModalGroup(
  MINSTANCE mInst,
  int group,
  double *val);
```

LUA Syntax:

```
val, rc = mc.mcCntlGetModalGroup(
  number mInst,
  number group)
```

Description: Get the modal group code for the specified group.

Parameters:

Parameter	Description
mInst	The controller instance.
group	A integer specifying the modal group.
val	The address of a double to receive the modal group value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**val** is NULL or **group** is out of range.

Notes:
If the modal group is not active, -1 is returned.

Usage:

```
// Get the modal code for modal group 1.
MINSTANCE mInst = 0;
int rc = mcCntlGetModalGroup(mInst, 1);
if (rc == MERROR_NOERROR) {
 // Success!
}
```

mcCntlGetMode

mcCntlGetOffset

C/C++ Syntax:

```
int mcCntlGetOffset(
  MINSTANCE mInst,
  int axisId,
  int type,
  double *offset);
```

LUA Syntax:

```
offset, rc = mc.mcCntlGetOffset(
  number mInst,
  number axisId,
  number type)
```

Description: Retrieve the offset specified by type for the axis specified by axisId

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	An Interger specifying the axis ID. (X_AXIS, Y_AXIS, Z_AXIS)
type	An Interger specifying the offset type. (MC_OFFSET_FIXTURE, MC_OFFSET_AXIS

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

Usage:

```
//
MINSTANCE mInst = 0;
```

mcCntlGetPoundVar

C/C++ Syntax:

```
int mcCntlGetPoundVar(
  MINSTANCE mInst,
  int param,
  double *value);
```

LUA Syntax:

```
value, rc = mc.mcCntlGetPoundVar(
  number mInst,
  number param)
```

Description: Get the value of a system variable (pound variable).

Parameters:

Parameter	Description
mInst	The controller instance.
param	An integer specifying a system variable.
*value	The address of a double to receive the value of the system variable.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**param** is out of range.

Notes:

This is a good tool to assist in debugging G code files with system variable (pound variable) programing.

Note that system variables can reflect values in the future because of the look ahead buffer.

Usage:

```
int mInst=0;
```

```
double PoundVar=50;
dobule Value=0;
;// Get the value of #50.
mcCntlGetPoundVar(mInst, PoundVar, &Value;);
```

mcCntlGetRegister

C/C++ Syntax:

LUA Syntax:

Description:

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

Usage:

```
//
MINSTANCE mInst = 0;
```

mcCntlGetRunTime

C/C++ Syntax:

```
int mcCntlGetRunTime(
  MINSTANCE mInst,
  double *time);
```

LUA Syntax:

```
int mcCntlGetRunTime(MINSTANCE mInst, double *time)
```

Description: Get the control run time in seconds.

Parameters:

Parameter	Description
mInst	The controller instance.
time	The address of a double to receive the run time in seconds.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**time** is NULL.

Notes:
None.

Usage:

```
// Get the control run time in seconds.
MINSTANCE mInst = 0;
double time;
int rc = mcCntlGetRunTime(mInst, &time;);
if (rc == MERROR_NOERROR) {
 // Success!
}
```

mcCntlGetSpindleSpeed

C/C++ Syntax:

```
mcCntlSetSpindleSpeed(
 MINSTANCE mInst,
 double secs);
```

Notes: Not used at this time

END_FUNTION

mcCntlGetState

C/C++ Syntax:

```
int mcCntlGetState(
  MINSTANCE mInst,
  mcState *state);
```

LUA Syntax:

```
mcState, rc = mc.mcCntlGetState(
  MINSTANCE mInst)
```

Description: Get the current controller state.

Parameters:

Parameter	Description
mInst	The controller instance.
state	The address of a mcState variable to receive the controller state.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**state** is NULL.

Notes:
None.

Usage:

```
// Get the state of controller instance 0.
MINSTANCE mInst = 0;
mcState state;
char stateName[80];
int rc = mcCntlGetState(mInst, &state;);
if (rc == MERROR_NOERROR) {
 // Success!
 rc = mcCntlGetStateName(mInst, state, stateName, sizeof(stateName));
}
```

mcCntlGetStateName

C/C++ Syntax:

```
int mcCntlGetStateName(
  MINSTANCE mInst,
  mcState state,
  char *buf,
  size_t bufSize);
```

LUA Syntax:

```
buf, rc = mc.mcCntlGetStateName(
  number mInst,
  number state)
```

Description: Retrieve the state name for the given mcState.

Parameters:

Parameter	Description
mInst	The controller instance.
state	A mcState variable specifying the state.
buf	A string buffer to receive the state name..
bufSize	The length of the string buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**buf** is NULL or **bufSize** is less than or equal to zero.
MERROR_INVALID_STATE	**state** is out of range.

Notes:
None.

Usage:

```
// Get the state name for controller instance 0.
```

```
MINSTANCE mInst = 0;
mcState state;
char stateName[80];
int rc = mcCntlGetState(mInst, &state;);
if (rc == MERROR_NOERROR) {
 // Success!
 rc = mcCntlGetStateName(mInst, state, stateName, sizeof(stateName));
}
```

mcCntlGetStats

C/C++ Syntax:

```
int mcCntlGetStats(
  MINSTANCE mInst,
  mstats_t *stats);
```

LUA Syntax:

```
N/A
```

Description: Retrieve the control statistics.

Parameters:

Parameter	Description
mInst	The controller instance.
stats	The address of a mstats_t variable to receive the statistics.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**stats** is NULL.

Notes:

```
struct mstats {
 int cannon_buf_depth;
 int la_cannon_buf_depth;
 double totalTime;
 double sessionTime;
 double spindleTime;
 long m3count;
 long m4count;
 long m6count;
 double xDistance;
 double yDistance;
 double zDistance;
 double aDistance;
 double bDistance;
```

```
 double cDistance;
};
typedef struct mstats mstats_t;
```

Usage:

```
// Get controller statistics.
MINSTANCE mInst = 0;
mstats_t stats;
int rc = mcCntlGetStats(mInst, &stats;);
if (rc == MERROR_NOERROR) {
 printf("M3 count = %dn", stats.m3Count);
}
```

mcCntlGetToolOffset

C/C++ Syntax:

```
int mcCntlGetToolOffset(
  MINSTANCE mInst,
  int axisId,
  double *offset);
```

LUA Syntax:

```
offset, rc = mc.mcCntlGetToolOffset(
  number mInst,
  number axisId)
```

Description: Get the tool offset distance for the specified axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	An integer specifying the axis.
offset	The address of a double that receives the offset distance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.
MERROR_INVALID_ARG	**offset** is NULL.

Notes:
None

Usage:

```
int mInst=0;
int toolnum = 5;
double val = 0;
```

```
mcCntlGetToolOffset(mInst, Z_AXIS, &val;); // Get the tool offset distance for tl
```

mcCntlGetUnitsCurrent

C/C++ Syntax:

```
int mcCntlGetUnitsCurrent(
  MINSTANCE mInst,
  int *units);
```

LUA Syntax:

```
units, rc = mc.mcCntlGetUnitsCurrent(
  number mInst)
```

Description: Get the current machine units.

Parameters:

Parameter	Description
mInst	The controller instance.
units	The address of an integer to receive the current machine units. (200 = inches

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**units** is NULL.

Notes:
None.

Usage:

```
// Get the current machine units.
MINSTANCE mInst = 0;
int units;
int rc = mcCntlGetUnitsCurrent(mInst, &units;);
if (rc == MERROR_NOERROR) {
 // Success!
}
```

mcCntlGetUnitsDefault

C/C++ Syntax:

```
int mcCntlGetUnitsDefault(
  MINSTANCE mInst,
  int *units);
```

LUA Syntax:

```
units, rc = mc.mcCntlGetUnitsDefault(
  number mInst)
```

Description: Get the default machine units.

Parameters:

Parameter	Description
mInst	The controller instance.
units	The address of an integer to receive the default machine units. (200 = inches

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**units** is NULL.

Notes:
None.

Usage:

```
// Get the default machine units.
MINSTANCE mInst = 0;
int units;
int rc = mcCntlGetUnitsDefault(mInst, &units;);
if (rc == MERROR_NOERROR) {
 // Success!
}
```

mcCntlGetValue

C/C++ Syntax:

```
int mcCntlGetValue(
  MINSTANCE  mInst,
  int valId,
  int param,
  double *value);
```

LUA Syntax:

```
value, rc = mc.mcCntlGetValue(
  number mInst,
  number valId,
  number param)
```

Description: Get the value of a core variable.

Parameters:

Parameter	Description
mInst	The controller instance.
valId	An integer specifying the core variable (VAL_ prefix in MachAPI.h).
param	An integer specifying the parameter for the core variable
value	The address of a double to receive the value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_IMPLEMENTED	The value specified by **valId** is not implemented.
various	Depending upon the value specified by **valId**.

Notes:
This is for ArtSoft's use. It is a generic interface for retrieving values from the core. Use of this API

function is dicouraged in favor of using one of the clearly named API calls to retrieve values for GUI and plugin programming.

Usage:

```
int mInst = 0;
int motor = 1;
double value = 0;
// Get the motor velocity of motor1.
int rc = mcCntlGetValue(mInst, VAL_MOTOR_VEL, motor, &value;);
```

mcCntlGetVersion

C/C++ Syntax:

```
int mcCntlGetVersion(
  MINSTANCE mInst,
  char *buf,
  size_t bufSize);
```

LUA Syntax:

```
buf, rc = mc.mcCntlGetVersion(
  number mInst)
```

Description: Retrieve the core version string.

Parameters:

Parameter	Description
mInst	The controller instance.
buf	A string buffer to receive the core version string.
bufSize	The length of the string buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**buf** is NULL or **bufSize** is 0.

Notes:
None.

Usage:

```
// Get the core version string.
MINSTANCE mInst = 0;
char ver[80];
int rc = mcCntlGetVersion(mInst, ver, sizeof(ver)):
if (rc == MERROR_NOERROR) {
 printf("The core version is %sn", ver);
}
```

mcCntlIsStill

C/C++ Syntax:

```
int mcCntlIsStill(
  MINSTANCE mInst,
  BOOL *still);
```

LUA Syntax:

```
still, rc = mcCntlIsStill(
  number mInst)
```

Description: Determine if all controlled axes are still.

Parameters:

Parameter	Description
mInst	The controller instance.
still	The address of a BOOL to receive the still state of the control.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**still** is NULL.

Notes:
This function cannot be used to determine if the control is finished processing G code. For example, the call may return TRUE if the control is in a dwell (G04).

Usage:

```
// See if the control axes are still.
MINSTANCE mInst = 0;
BOOL still;
int rc = mcCntlIsStill(mInst, &still;);
if (rc == MERROR_NOERROR) {
 if (still == TURE) {
  // All axes are still.
 } else {
```

```
  // At least one axis is moving!
 }
}
```

mcCntlMachineStateClear

C/C++ Syntax:

```
int mcCntlMachineStateClear(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcCntlMachineStateClear(
  number mInst)
```

Description: Clears the machine control state stack.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
This function is used to clear a stack of control states and is primarily used in a scripting environment. Control states are pushed on a stack with mcCntlMachineStatePush().

Usage:

```
-- LUA example
function test()
 local inst = mc.mcGetInstance();
 -- push control state to the stack.
 mc.mcCntlMachineStatePush(inst);
 -- put machine in G20, and G90 mode.
 mc.mcCntlGcodeExecuteWait(inst, "G20nG90");
 -- push control state to the stack.
 mc.mcCntlMachineStatePush(inst);
 -- put machine in G21, and G91 mode.
 mc.mcCntlGcodeExecuteWait(inst, "G21nG91");
 -- clear the machine state stack and leave the machine in G21 and G91 modes.
 mc.mcCntlMachineStateClear(inst);
end
```

mcCntlMachineStatePop

C/C++ Syntax:

```
int mcCntlMachineStatePop(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcCntlMachineStatePop(
  number mInst)
```

Description: Pops a machine control state off the stack.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
This function is used to return the control to a state previously saved on the state stack and is primarily used in a scripting environment. Control states are pushed (saved) on a stack with mcCntlMachineStatePush().

Usage:

```
-- LUA example
function test()
-- LUA example
function test()
 local inst = mc.mcGetInstance();
 -- push control state to the stack saving original modes.
 mc.mcCntlMachineStatePush(inst);
 -- put machine in G20, and G90 mode.
 mc.mcCntlGcodeExecuteWait(inst, "G20nG90");
 -- push control state to the stack.
 mc.mcCntlMachineStatePush(inst);
 -- put machine in G21, and G91 mode.
 mc.mcCntlGcodeExecuteWait(inst, "G21nG91");
```

```
 -- restore the machine state stack to G20 and G90 modes.
 mc.mcCntlMachineStatePop(inst);
 -- restore the machine state stack to original modes.
 mc.mcCntlMachineStatePop(inst);
end
```

mcCntlMachineStatePush

C/C++ Syntax:

```
int mcCntlMachineStatePush(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcCntlMachineStatePush(
  number mInst)
```

Description: Pushes the machine control state on the stack.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
This function is used to save the control state to a stack of states and is primarily used in a scripting environment.

Usage:

```
-- LUA example
function test()
 local inst = mc.mcGetInstance();
 -- push control state to the stack saving original modes.
 mc.mcCntlMachineStatePush(inst);
 -- put machine in G20, and G90 mode.
 mc.mcCntlGcodeExecuteWait(inst, "G20nG90");
 -- push control state to the stack.
 mc.mcCntlMachineStatePush(inst);
 -- put machine in G21, and G91 mode.
 mc.mcCntlGcodeExecuteWait(inst, "G21nG91");
 -- restore the machine state stack to G20 and G90 modes.
 mc.mcCntlMachineStatePop(inst);
 -- restore the machine state stack to original modes.
```

```
 mc.mcCntlMachineStatePop(inst);
end
```

mcCntlMdiExecute

C/C++ Syntax:

```
int mcCntlMdiExecute(
  MINSTANCE mInst,
  const char *commands);
```

LUA Syntax:

```
rc = mc.mcCntlMdiExecute(
  number mInst,
  string commands)
```

Description: Used to execute G code commands that are not part of the main G code execution.

Parameters:

Parameter	Description
mInst	The controller instance.
commands	A string buffer with the line or lines of Gcode.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**commands** is NULL.

Notes:
Used to run commands like MDI or script commands. The function returns immediately without waiting on the G code to proocess. It is an error to call this function multiple times for each block of G code. Multiple blocks should be concatenated and separated with new line characters to form a "unit of work".

Usage:

```
MINSTANCE mInst = 0;
int rc;
// Move the machine back to xy then z zero.
// Correct example.
rc = mcCntlMdiExecute(mInst, "G00 G90 X0.0 Y0.0\nZ0.0");
```

```
// Incorrect example.
rc = mcCntlMdiExecute(mInst, "G00 G90 X0.0 Y0.0");
rc = mcCntlMdiExecute(mInst, "Z0.0");
```

mcCntlPluginConfig

C/C++ Syntax:

LUA Syntax:

Description:

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

Usage:

```
//
MINSTANCE mInst = 0;
```

mcCntlPluginDiag

C/C++ Syntax:

LUA Syntax:

Description:

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

Usage:

```
//
MINSTANCE mInst = 0;
```

mcCntlProbeFileClose

C/C++ Syntax:

```
int mcCntlProbeFileClose(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcCntlProbeFileClose(
  number mInst)
```

Description: Close the probe data file.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Closes a probe data file that was previously opened with mcCntlProbeFileOpen(). This function is primarily used in a scripting environment.

Usage:

```
-- probe file close LUA macro example.
function m112()
 local inst=mc.mcGetInstance();
 local rc = mc.mcCntlProbeFileClose(inst);
end

if (mc.mcInEditor() == 1) then
 m112()
end
```

mcCntlProbeFileOpen

C/C++ Syntax:

```
int mcCntlProbeFileOpen(
  MINSTANCE mInst,
  const char *fileName,
  const char *format,
  BOOL overWrite);
```

LUA Syntax:

```
rc = mc.mcCntlProbeFileOpen(
  number mInst,
  string fileName,
  string format,
  number overWrite);
```

Description: Open a probe data collection file.

Parameters:

Parameter	Description
mInst	The controller instance.
fileName	A string buffer containing the full path of the probe data file to be opened.
format	A string buffer containing the format description for the probe data file.
overWrite	A BOOL specifying if the file should be over written if it exists..

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

Opens a probe data collection file and specifies the format in which the data is saved. This function is primarily used in a scripting environment.

The **format** parameter uses subtitutions. Numbers are formatted with the C/C++ printf style notation. Axis values are substituted with AXIS_X, AXIS_Y, AXIS_Z, AXIS_A, AXIS_B, and AXIS_C.

Usage:

```
-- probe file open LUA macro example.
function m111()
 local inst = mc.mcGetInstance();
 local rc = mc.mcCntlProbeFileOpen(inst, 'probetest.csv', '%.4AXIS_X, %.4AXIS_Y,
end

if (mc.mcInEditor() == 1) then
 m111()
end
```

mcCntlSetCoolantDelay

C/C++ Syntax:

```
int mcCntlSetCoolantDelay(
  MINSTANCE mInst,
  double secs);
```

LUA Syntax:

```
rc = mc.mcCntlSetCoolantDelay(
  number mInst,
  number secs);
```

Description: Set the coolant delay in seconds.

Parameters:

Parameter	Description
mInst	The controller instance.
seconds	A double specifying the delay in seconds.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Seconds can be a decimal value. e.g. .5 for half of a second.

Usage:

```
// Set 3/4 second coolant delay.
MINSTANCE mInst = 0;
int rc = mcCntlSetCoolantDelay(mInst, .750);
```

mcCntlSetDiaMode

C/C++ Syntax:

```
int mcCntlSetDiaMode(
  MINSTANCE mInst,
  BOOL enable);
```

LUA Syntax:

```
rc = mc.mcCntlSetDiaMode(
  number mInst,
  number enable);
```

Description:

Parameters:

Parameter	Description
mInst	The controller instance.
enable	A BOOL specifying the siameter mode state. (TRUE/FALSE)

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

This function is only valid when the control is in lathe mode.

Usage:

```
// Enable diamter mode
MINSTANCE mInst = 0;
int rc = mcCntlSetDiaMode(mInst, TRUE);
```

mcCntlSetMistDelay

C/C++ Syntax:

```
int mcCntlSetMistDelay(
  MINSTANCE mInst,
  double secs);
```

LUA Syntax:

```
rc = mc.mcCntlSetMistDelay(
  number mInst,
  number secs);
```

Description: Set the delay for the mist coolant in seconds.

Parameters:

Parameter	Description
mInst	The controller instance.
secs	A double specifying the delay in seconds.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Decimal values are allowed. e.g. .5 is 1/2 of a second.

Usage:

```
// Set the mist delay to 3/4 of a second.
MINSTANCE mInst = 0;
int rc = mcCntlSetMistDelay(mInst, .750);
```

mcCntlSetMode

C/C++ Syntax:

```
int mcCntlSetMode(
  MINSTANCE mInst,
  double mode);
```

LUA Syntax:

```
rc = mc.mcCntlSetMode(
  number mInst,
  number mode)
```

Description: Set the interpreter mode of the controller instance.

Parameters:

Parameter	Description
mInst	The controller instance.
mode	the interpreter mode of the controller. (MC_MODE_MILL, MC_MODE_LATHE_DIA, MC_MODE_LATHE_RAD, MC_MODE_TANGENTIAL)

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
The core needs to be restarted if the mode is changed.

Usage:

```
// Set the interpreter mode to Lathe Diameter.
MINSTANCE mInst = 0;
int rc = mcCntlSetMode(mInst, MC_MODE_LATHE_DIA);
```

mcCntlSetPoundVar

C/C++ Syntax:

```
mcCntlSetPoundVar(
  MINSTANCE mInst,
  int param,
  double value);
```

LUA Syntax:

```
mcCntlSetPoundVar(
  number mInst,
  number param,
  number value)
```

Description: Set the value of a system variable (pound variable).

Parameters:

Parameter	Description
mInst	The controller instance.
param	system variable index.
value	The value to apply to the system variable (pound variable).

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**param** is out of range.

Notes:
This function should only be used when G code is not processing.

Usage:

```
MINSTANCE mInst = 0;
int PoundVar = 50;
dobule Value=21;
// Set the value of #50 to 21.
int rc = mcCntlSetPoundVar(mInst, PoundVar, Value);
```

mcCntlSetStats

C/C++ Syntax:

```
int mcCntlSetStats(
  MINSTANCE mInst,
  mstats_t *stats);
```

LUA Syntax:

```
N/A
```

Description: Set/reset the machine statistics.

Parameters: (stats, The address of a mstats_t struct with which to set the control statistics.

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**stats** is NULL.

Notes:

Usage:

```
//
MINSTANCE mInst = 0;
```

mcCntlSetValue

C/C++ Syntax:

```
int mcCntlSetValue(
  MINSTANCE  mInst,
  int valId,
  int param,
  double value);
```

LUA Syntax:

```
rc = mc.mcCntlSetValue(
  number mInst,
  number valId,
  number param,
  number value)
```

Description: Set the value of a control variable.

Parameters:

Parameter	Description
mInst	The controller instance.
valId	The control variable to set (VAL_ prefix in MachAPI.h).
param	The parameter for the control value (or 0).
value	The new value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_IMPLEMENTED	The value specified by **valId** is not implemented.
Various	Depending upon **valId**.

Notes:
This function is used by Artsoft to add control values to the core on a per OEM basis. Use of this function is discouraged in favor of using the more clearly named API functions for GUI and plugin

programming.

Usage:

```
MINSTANCE mInst = 0;
int motor = 1;
value=1000.0;
// Set the Velocity of the motor to 1000.0 .
in rc = mcCntlSetValue(mInst, VAL_MOTOR_VEL, motor, value);
```

mcFileHoldAquire

C/C++ Syntax:

```
int mcFileHoldAquire(
  MINSTANCE mInst,
  const char *reason,
  int JogAxisBits);
```

LUA Syntax:

```
rc = mc.mcFileHoldAquire(
  number mInst,
  string reason,
  number JogAxisBits)
```

Description: Used to hold G code processing.

Parameters:

Parameter	Description
mInst	The controller instance.
reason	A string buffer specifying the reason for the file hold.
JogAxisBits	A bitmask of bits specifying the axes that are allowed to jog when in then file hold state.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_STATE	The control is not in a state where file hold is allowed.

Notes:
This function is primarily used in a scripting environment.

Usage:

```
-- LUA example
local inst = mc.mcGetInstance();
local rc = mc.mcFileHoldAquire(inst, "My hold reason", 0);
```

mcFileHoldReason

C/C++ Syntax:

```
int mcFileHoldReason(
  MINSTANCE mInst,
  char *buf,
  long bufSize);
```

LUA Syntax:

```
reason, rc = mc.mcFileHoldReason(
  number mInst)
```

Description: Returns a string clarifying why the file processing was held.

Parameters:

Parameter	Description
mInst	The controller instance.
buf	A string buffer that receives the reason the file was held.
bufSize	A long specifying the length of the string buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_STATE	The control is not in the file hold state.
MERROR_INVALIS_ARG	**buf** is NULL or **bufSize** is less than or equal to 0.

Notes:
This function is primarily used in a scripting environment.

Usage:

```
-- LUA example
local inst = mc.mcGetInstance();
local rc = mc.mcFileHoldAquire(inst, "My hold reason", 0);
loacl reason
```

```
reason, rc = mc.mcFileHoldReason(inst);
```

mcFileHoldRelease

C/C++ Syntax:

```
int mcFileHoldRelease(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcFileHoldRelease(
  number mInst)
```

Description: Used to exit a file hold state.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_STATE	The control was not in the file hold state.

Notes:
This function is primarily used in a scripting environment.

Usage:

```
-- LUA example
local inst = mc.mcGetInstance();
local rc = mc.mcFileHoldAquire(inst, "My hold reason", 0);
loacl reason
reason, rc = mc.mcFileHoldReason(inst);
-- Do some script magic...
rc = mc.mcFileHoldRelease(inst);
-- G code processing resumes.
```

Gcode File

mcCntlGetGcodeLine

C/C++ Syntax:

```
int mcCntlGetGcodeLine(
  MINSTANCE mInst,
  int LineNumber,
  char *buf,
  long bufSize);
```

LUA Syntax:

```
buff, rc = mc.mcCntlGetGcodeLine(
  number mInst,
  number LineNumber)
```

Description: Get the line of Gcode from the loaded gcode file.

Parameters:

Parameter	Description
mInst	The controller instance.
LineNumber	The desired line number from the Gcode file.
buf	The address of a character buffer that recieves the Gcode line.
bufSize	The size of the receiving character buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**LineNumber** is out of range

Notes:
Used to get the gocde file lines to display

Usage:

```
int mInst=0;
int LineNumber = 5;
char gline[128];
```

```
gline[0] = '0';
mcCntlGetGcodeLine(mInst, LineNumber, gline , 128); // Get line number 5 from the
```

mcCntlGetGcodeLineCount

C/C++ Syntax:

```
int mcCntlGetGcodeLineCount(
  MINSTANCE mInst,
  double *count);
```

LUA Syntax:

```
count, rc = mc.mcCntlGetGcodeLineCount(
  number mInst)
```

Description: Get the number of lines in the Gcode file.

Parameters:

Parameter	Description
mInst	The controller instance.
count	The address of a double to receive the number of lines in the G code file.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
The number of lines in the file may not be the number of blocks or moves to be done because of sub routines and canned cycles.

Usage:

```
int mInst=0;
double count=0;
mcCntlGetGcodeLineCount(mInst, &count;);
```

mcCntlGetGcodeLineNbr

C/C++ Syntax:

```
int mcCntlGetGcodeLineNbr(
  MINSTANCE mInst,
  double *val);
```

LUA Syntax:

```
val, rc = mcCntlGetGcodeLineNbr(
  number mInst)
```

Description: Returns the line number of the current move being made.

Parameters:

Parameter	Description
mInst	The controller instance.
val	The address of a double to receive the G code line number.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
This function will return the next line that will be run if G code is not running.

Usage:

```
int mInst=0;
double dline=0;
mcCntlGetGcodeLineNbr(mInst, &dline;); //Get the current running line number
```

mcCntlRewindFile

C/C++ Syntax:

```
int mcCntlRewindFile(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcCntlRewindFile(
  number mInst)
```

Description: Rewind the G code file.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_NOW	The G code file is running and not allowing a rewind.

Notes:
A MSG_REWIND_FILE notification is sent to the GUI and plugins.

Usage:

```
MINSTANCE mInst = 0;
mcCntlRewindFile(mInst); // Rewind controller 0's Gcode file.
```

mcCntlSetGcodeLineNbr

C/C++ Syntax:

```
int mcCntlSetGcodeLineNbr(
  MINSTANCE mInst,
  double line);
```

Description: Set the Gcode line number to run / start.

Parameters:

Parameter	Description
mInst	The controller instance.
line	Sets the current line in the Gcode file.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
This will not update the running state engine so great care should be used when setting the line.
Do not set the line when processing G code.
Calling this function sends MSG_GCODE_LINE to the GUI and plugins.
The parameter **line** is base 1.

Usage:

```
int mInst=0;
mcCntlSetGcodeLineNbr(mInst, 10); // Set the Gcode file to be at line 10.
```

mcCntlLoadGcodeFile

C/C++ Syntax:

```
int mcCntlLoadGcodeFile(
  MINSTANCE mInst,
  const char *FileToLoad);
```

LUA Syntax:

```
rc = mc.mcCntlLoadGcodeFile(
  number mInst,
  stringFileToLoad)
```

Description: Load a G code file.

Parameters:

Parameter	Description
mInst	The controller instance.
FileToLoad	A string buffer containing the Gcode file name.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**FileToLoad** is NULL.
MERROR_FILE_EXCEPTION	Exception while loading file.
MERROR_FILE_EMPTY	No data a in file.
MERROR_FILE_SHARING	Violation sharing file.
MERROR_FILE_INVALID	Not a valid file type.
MERROR_FILE_BADSIZE	Invalid file size (over 1.5GB).
MERROR_FILE_NOT_FOUND	The file cannot be found in the file system.
MERROR_FILE_BADFORMAT	The file had a line end without a new line character. The file may be incomplete.

Notes:
Used to load a file into a controller instance.

Usage:

```
int mInst=0;
char File[128] = "C:SomeGocdefile.tap";
int rc = mcCntlLoadGcodeFile(mInst, &char;); // Load SomeGocdefile.tap file.
if (rc == MERROR_NOERROR) {
 // Success!
}
```

mcCntlLoadGcodeString

C/C++ Syntax:

```
int mcCntlLoadGcodeString(
  MINSTANCE mInst,
  const char *gCode);
```

LUA Syntax:

```
rc = mc.mcCntlLoadGcodeString(
  MINSTANCE mInst,
  const char *gCode);
```

Description: Loads G code from a string.

Parameters:

Parameter	Description
mInst	The controller instance.
gCode	A string buffer containing G code.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**gCode** is NULL.

Notes:
This function will load G code from a string instead of a file. It is primarily used for static test or repeating operations.

Usage:

```
// Load G code from a string.
MINSTANCE mInst = 0;
char *gCode = "%nO1001nG90 G94 G91.1 G40 G49 G17nG20"
int rc = mcCntlLoadGcodeString(mInst, gCode);
```

mcCntlCloseGCodeFile

C/C++ Syntax:

```
int mcCntlCloseGCodeFile(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcCntlCloseGCodeFile(
  number mInst)
```

Description: Close Gocde file.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Used to close a Gcode file if the Gcode File needs to be edited

Usage:

```
int mInst=0;
mcCntlCloseGCodeFile(mInst); // Close the Gcode file in controller 0.
```

mcCntlGetGcodeFileName

C/C++ Syntax:

```
int mcCntlGetGcodeFileName(
  MINSTANCE mInst,
  char *buf,
  size_t bufSize);
```

LUA Syntax:

```
buf, rc = mc.mcCntlGetGcodeFileName(
  number mInst)
```

Description: Retrieve he current G code file name.

Parameters:

Parameter	Description
mInst	The controller instance.
buf	A string buffere to receive the G code file name.
bufSize	The length of the string buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
An empty string is returned if there is no G code file loaded.

Usage:

```
// Get the currently loaded G code file name.
MINSTANCE mInst = 0;
char gCodeFileName[255];
int rc = mcCntlGetGcodeFileName(mInst, gCodeFileName, sizeof(gCodeFileName));
if (rc == MERROR_NOERROR) {
 // Success.
}
```

Tools

mcToolGetCurrent

C/C++ Syntax:

```
int mcToolGetCurrent(
  MINSTANCE mInst,
  int *toolnum);
```

LUA Syntax:

```
toolnum, rc = mc.mcToolGetCurrent(
  number mInst)
```

Description: Retrieve the current tool number.

Parameters:

Parameter	Description
mInst	The controller instance.
toolnum	The address of an integer to receive the current tool number.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Tool numbers start at 1 (base 1).

Usage:

```
// Get the current tool.
MINSTANCE mInst = 0;
int toolnum = 0;
int rc = mcToolGetCurrent(mInst, &toolnum;);
```

mcToolGetData

C/C++ Syntax:

```
int mcToolGetData(
  MINSTANCE mInst,
  int type,
  int toolNumber,
  double *value);
```

LUA Syntax:

```
value, rc = mc.mcToolGetData(
  number mInst,
  number type,
  number toolNumber)
```

Description: Retrieve the offset values for mill and turn tools.

Parameters:

Parameter	Description
mInst	The controller instance.
type	MTOOL_MILL_X, MTOOL_MILL_X_W, MTOOL_MILL_Y, MTOOL_MILL_Y_W, MTOOL_MILL_HEIGHT, MTOOL_MILL_HEIGHT_W, MTOOL_MILL_RAD, MTOOL_MILL_RAD_W, MTOOL_MILL_POCKET, MTOOL_LATHE_X, MTOOL_LATHE_X_W, MTOOL_LATHE_Y, MTOOL_LATHE_Y_W, MTOOL_LATHE_Z, MTOOL_LATHE_Z_W, MTOOL_LATHE_POCKET, MTOOL_LATHE_TIPRAD, MTOOL_MILL_RAD, MTOOL_LATHE_TIPDIR, MTOOL_MILL_RAD_W, MTOOL_LATHE_TOOLSLIDE, MTOOL_TYPE
toolNumber	Specifies the tool number.
value	The address of a double to receive the value for the specified offset type and the specified tool.

Returns:

Return Code	Description

MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**toolNumber** is out of range.

Notes:
Used to get the offset of a tool and can be used at anytime.

Usage:

```
MINSTANCE mInst = 0;
int toolnum = 5;
double val = 0;
// Get the tool height offset for tool number 5.
int rc = mcToolGetData(mInst, MTOOL_MILL_HEIGHT, toolnum, &val;);
```

mcToolGetDesc

C/C++ Syntax:

```
int mcToolGetDesc(
  MINSTANCE mInst,
  int toolnum,
  char *buff,
  size_t bufsize);
```

LUA Syntax:

```
buff, rc = mc.mcToolGetDesc(
  number mInst,
  number toolnum)
```

Description: Get the text string to describe the tool

Parameters:

Parameter	Description
mInst	The controller instance.
toolnum	An integer specifying the tool number for which to get the description.
buff	The address of a string buffer to receive the tool description.
bufsize	The length of the string buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**toolnum** is out of range.

Notes:
Used to get the tool description for the tool table or run screen.

Usage:

```
MINSTANCE mInst = 0;
char desc[128];
```

```
int toolnum = 5;
// Get the description of tool number 5 for controller 0.
int rc = mcToolGetDesc(mInst, toolnum, desc, sizeof(desc));
```

mcToolGetSelected

C/C++ Syntax:

```
int mcToolGetSelected(
  MINSTANCE mInst,
  int *toolnum);
```

LUA Syntax:

```
toolnum, rc = mc.mcToolGetSelected(
  number mInst)
```

Description: Retrieve the selected (next) tool number.

Parameters:

Parameter	Description
mInst	The controller instance.
toolnum	The address of an integer to receive the selected tool number.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Get the selected tool number.
MINSTANCE mInst = 0;
int toolnum = 0;
int rc = mcToolGetSelected(mInst, &toolnum;);
```

mcToolLoadFile

C/C++ Syntax:

```
int mcToolLoadFile(
  MINSTANCE mInst,
  const char *FileToLoad);
```

LUA Syntax:

```
rc = mc.mcToolLoadFile(
  number mInst,
  string FileToLoad)
```

Description: Load a tool table into the core.

Parameters:

Parameter	Description
mInst	The controller instance.
FileToLoad	A string buffer specifying the tool table file to load.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_FILE_EMPTY	The specified tool table file was empty.

Notes:
Used to load a tool table file on a per profile basis.

Usage:

```
// Load a tool table.
MINSTANCE mInst = 0;
int rc = mcToolLoadFile(mInst, "tooltable.tls");
```

mcToolSaveFile

C/C++ Syntax:

```
int mcToolSaveFile(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc =  mcToolSaveFile(
  number mInst)
```

Description: Save a previously loaded tool table file.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_FILE_INVALID	No tool table fie was previously loaded.

Notes:
None.

Usage:

```
// Save the previously loaded tool table file.
MINSTANCE mInst = 0;
int rc = mcToolSaveFile(mInst);
```

mcToolSetCurrent

C/C++ Syntax:

```
int mcToolSetCurrent(
  MINSTANCE mInst,
  int toolnum);
```

LUA Syntax:

```
rc = mc.mcToolSetCurrent(
  number mInst,
  number toolnum)
```

Description: Set the current tool to the specified tool.

Parameters:

Parameter	Description
mInst	The controller instance.
toolnum	An integer specifying the tool.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**toolnum** is out of range.

Notes:
None.

Usage:

```
// Set the current tool to tool #1.
MINSTANCE mInst = 0;
int rc = mcToolSetCurrent(mInst, 1);
```

mcToolSetData

C/C++ Syntax:

```
int mcToolSetData(
  MINSTANCE mInst,
  int Type,
  int Toolnumber,
  double val);
```

LUA Syntax:

```
rc = mc.mcToolSetData(
  number mInst,
  number Type,
  number Toolnumber,
  number val);
```

Description: Set the offset values for each tool.

Parameters:

Parameter	Description
mInst	The controller instance.
Type	MTOOL_MILL_X, MTOOL_MILL_X_W, MTOOL_MILL_Y, MTOOL_MILL_Y_W, MTOOL_MILL_HEIGHT, MTOOL_MILL_HEIGHT_W, MTOOL_MILL_RAD, MTOOL_MILL_RAD_W, MTOOL_MILL_POCKET, MTOOL_LATHE_X, MTOOL_LATHE_X_W, MTOOL_LATHE_Y, MTOOL_LATHE_Y_W, MTOOL_LATHE_Z, MTOOL_LATHE_Z_W, MTOOL_LATHE_POCKET, MTOOL_LATHE_TIPRAD, MTOOL_MILL_RAD, MTOOL_LATHE_TIPDIR
Toolnumber	Tool number to be set
val	Value to set the selected offset.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.

MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**Toolnumber** is out of range.

Notes:
Used to set the offest of a tool and shouldn't be changed as Gcode is running.

Usage:

```
MINSTANCE mInst = 0;
int toolnum = 5;
double val = 0;
// Set the tool height wear offset to zero.
int rc = mcToolSetData(mInst, MTOOL_MILL_HEIGHT_W, toolnum, val);
```

mcToolSetDesc

C/C++ Syntax:

```
int mcToolSetDesc(
  MINSTANCE mInst,
  int toolnum,
  const char *tdsc);
```

LUA Syntax:

```
rc = mc.mcToolSetDesc(
  number mInst,
  number toolnum,
  string tdsc);
```

Description: Set the description for the specified tool in the tool table.

Parameters:

Parameter	Description
mInst	The controller instance.
toolnum	Tool number to which to set the description.
tdsc	A string buffer that specifies the description of the tool.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**toolnumber** is out of range.

Notes:
Used to set the description of a tool for the tool table or it can be used to allow the user to change it on the program run page.

Usage:

```
MINSTANCE mInst = 0;
char desc[128] = "My Best 1/2 inch endmill";
int toolnum = 5;
// Set the description of tool number 5.
int rc = mcToolSetDesc(mInst, toolnum, desc);
```

Jogging

- mcJogGetAccel
- mcJogGetInc
- mcJogGetRate
- mcJogGetVelocity
- mcJogIncStart
- mcJogIncStop
- mcJogIsJogging
- mcJogSetAccel
- mcJogSetInc
- mcJogSetRate
- mcJogSetTraceEnable
- mcJogSetType
- mcJogVelocityStart
- mcJogVelocityStop

mcJogGetAccel

C/C++ Syntax:

```
int mcJogGetAccel(
  MINSTANCE mInst,
  int axisId,
  double *percent);
```

LUA Syntax:

```
percent, rc = mc.mcJogGetAccel(
  number mInst,
  number axis)
```

Description: Get the current jog acceleration value for the give aixs as a percentage of max velocity.

Parameters: (percent, The address of a double to receive the current jog acceleration percentage.

Parameter	Description
mInst	The controller instance.
axisId	An integer specifying the axis.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.
MERROR_INVALID_ARG	**percent** is NULL.

Notes:

Usage:

```
// Get the jog accel percentage for the X axis.
MINSTANCE mInst = 0;
double accel;
int rc = mcJogGetAccel(mInst, X_AXIS, &accel;);
```

mcJogGetInc

C/C++ Syntax:

```
int mcJogGetInc(
  MINSTANCE mInst,
  int axisId,
  double *increment);
```

LUA Syntax:

```
increment, rc = mc.mcJogGetInc(
  number mInst,
  number axisId)
```

Description: Returns the current jog increment for the given axis.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.
MERROR_INVALID_ARG	**increment** is NULL.

Notes:

Usage:

```
//
MINSTANCE mInst = 0;
```

mcJogGetRate

C/C++ Syntax:

```
int mcJogGetRate(
  MINSTANCE mInst,
  int axisId,
  double *percent);
```

LUA Syntax:

```
percent, rc = mc.mcJogGetRate(
  number mInst,
  number axisId)
```

Description: Returns the current jog rate as a percentage of max velocity.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	An integer specifying the axis.
percent	The address of a double to receive the jog rate percentage.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.
MERROR_INVALID_ARG	**percentage** is NULL.

Notes:

Usage:

```
// Get the current jog rate for the X axis
MINSTANCE mInst = 0;
double jogRate;
int rc = mcJogGetRate(mInst, X_AXIS, &jogRate;);
```

mcJogGetVelocity

C/C++ Syntax:

```
int mcJogGetVelocity(
  MINSTANCE mInst,
  int axisId,
  double *vel);
```

LUA Syntax:

```
vel, rc = mc.mcJogGetVelocity(
  number mInst,
  number axisId),
```

Description: Get the current jog velocity setting in units per minute.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis from which to get the jog velocity.
vel	The address of a double to receive the current jog velocity.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Changing the jog rate (percentage of max velocity) will affect the return value.

Usage:

```
// Get the current jog velocity setting for the X axis
MINSTANCE mInst = 0;
double jogVel;
int rc = mcJogGetVelocity(mInst, X_AXIS, &jogVel;);
```

mcJogIncStart

C/C++ Syntax:

```
int mcJogIncStart(
  MINSTANCE mInst,
  int axisId,
  double dist);
```

LUA Syntax:

```
rc = mc.mcJogIncStart(
  number mInst,
  number axisId,
  number dist)
```

Description: Start an Incermetnal jog move.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis to start jogging.
dist	The incremental distance to jog the axis (added to current goal position).

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Inc jog or position any axis. The axis will use the MaxVel and Accel settings to get into position.

Usage:

```
MINSTANCE mInst = 0;
int axis = Z_AXIS;
double Joginc = .100;
// Jog controller 0 .1 in the Z axis.
```

```
mcJogIncStart(mInst, axis, Joginc);
```

mcJogIncStop

C/C++ Syntax:

```
int mcJogIncStop(
  MINSTANCE mInst,
  int axisId,
  double incr);
```

LUA Syntax:

```
rc = mc.mcJogIncStop(
  number mInst,
  number axisId,
  number incr)
```

Description: Stop an incremental jog at the closest increment.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis to stop jogging.
inc	The increment on which to to stop the axis.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
This function will stop an axis on an increment. This feature is like a digital detent to allow an axis to stop at some set increment. The original use is to stop an axis that is over loaded with inc jog requests.

Usage:

```
// Stop the incremental jog on the Z axis.
MINSTANCE mInst = 0;
```

```
int axis = Z_AXIS;
double Joginc = .100;
int rc = mcJogIncStop(mInst, axis, Joginc);
```

mcJogIsJogging

C/C++ Syntax:

```
int mcJogIsJogging(
  MINSTANCE mInst,
  int axisId,
  BOOL *jogging);
```

LUA Syntax:

```
jogging, rc = mc.mcJogIsJogging(
  number mInst,
  number axisId)
```

Description: Determine if the given axis is jogging.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	An integer specifying the axis.
jogging	The address of a BOOL to receive the axis jog state.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
None.

Usage:

```
// See if the X axis is jogging.
MINSTANCE mInst = 0;
BOOL jogging = FALSE;
int rc = mcJogIsJogging(mInst, X_AXIS, &jogging;);
```

mcJogSetAccel

C/C++ Syntax:

```
int mcJogSetAccel(
  MINSTANCE mInst,
  int axisId,
  double percent);
```

LUA Syntax:

```
rc = mc.mcJogSetAccel(
  number mInst,
  number axisId,
  double percent)
```

Description: Set the jog accel as a percentage of the axis' maximum velocity.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	An integer spcifying the axis.
percent	A double specifying the acceleration percentage.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:

Usage:

```
// Set the X axis jog accel percentage to 75%.
MINSTANCE mInst = 0;
int mcJogSetAccel(mInst, X_AXIS, 75);
```

mcJogSetInc

C/C++ Syntax:

```
int mcJogSetInc(
  MINSTANCE mInst,
  int axisId,
  double increment);
```

LUA Syntax:

```
rc = mc.mcJogSetInc(
  number mInst,
  number axisId,
  number increment)
```

Description: Set the current jog increment for the give axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	An integer specifying the axis.
increment	A double specifying the desired increment.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
None.

Usage:

```
// Set the X axis jog increment to .010"
MINSTANCE mInst = 0;
int rc = mcJogSetInc(mInst, X_AXIS, .010);
```

mcJogSetRate

C/C++ Syntax:

```
int mcJogSetRate(
  MINSTANCE mInst,
  int axisId,
  double percent);
```

LUA Syntax:

```
rc = mc.mcJogSetRate(
  number mInst,
  number axisId,
  number percent)
```

Description: Set the jog rate of the given axis as a percentage of the axis' maximum velocity.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis to stop jogging.
percent	The jog rate percentage.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
None

Usage:

```
// Set the jog rate to 75% of the Z axis maximum velocity.
MINSTANCE mInst = 0;
int axis = Z_AXIS;
int rc = mcJogSetRate(mInst, axis, 75);
```

mcJogSetTraceEnable

C/C++ Syntax:

```
int mcJogSetTraceEnable(
  MINSTANCE mInst,
  BOOL enable);
```

LUA Syntax:

```
rc = mc.mcJogSetTraceEnable(
  number mInst,
  number enable)
```

Description: Enable or disable tool path jog tracing.

Parameters:

Parameter	Description
mInst	The controller instance.
enable	A BOOL secifying tool path jog tracing.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None

Usage:

```
//Set jog tracing.
MINSTANCE mInst = 0;
int rc = mcJogSetTraceEnable(mInst, TRUE);
```

mcJogSetType

C/C++ Syntax:

```
int mcJogSetType(MINSTANCE mInst, int axisId, int type);
```

LUA Syntax:

```
rc = mc.mcJogSetType(
  number mInst,
  number axisId,
  number type)
```

Description: Set the jog operation type for the given axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	An integer specifying the axis.
type	An integer specifying the jog type. (MC_JOG_TYPE_VEL or MC_JOG_TYPE_INC)

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
None

Usage:

```
// Set the X axis jog type to velocity mode.
MINSTANCE mInst = 0;
int rc = mcJogSetType(mInst, X_AXIS, MC_JOG_TYPE_VEL);
```

mcJogVelocityStart

C/C++ Syntax:

```
int mcJogVelocityStart(
  MINSTANCE mInst,
  int axisId,
  double dir);
```

LUA Syntax:

```
rc = mc.mcJogVelocityStart(
  number mInst,
  number axisId,
  number dir);
```

Description: Start a velocity jog on the given axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axis	The axis to jog.
dir	An integer specifying the direction of the jog. (MC_JOG_POS, MC_JOG_NEG, MC_JOG_STOP)

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.
MERROR_NOT_NOW	The operation could not be completed at this time.

Notes:
Specifying MC_JOG_STOP for **dir** is the same as calling mcJogVelocityStop().

Usage:

```
int mInst=0;
```

```
int axis = Z_AXIS;
int rc = mcJogVelocityStart(mInst, axis, MC_JOG_POS); // Start Z axis jogging fo
if (rc == MERROR_NOERROR) {
 Sleep(5000);
}
rc = mcJogVelocityStart(mInst, axis, MC_JOG_POS); // Reverse axis and jog the ax
if (rc == MERROR_NOERROR) {
 Sleep(5000);
}
mcJogVelocityStop(mInst, axis); // Stop the axis.
```

mcJogVelocityStop

C/C++ Syntax:

```
mcJogVelocityStop(
  MINSTANCE mInst,
  int axisId);
```

LUA Syntax:

```
mcJogVelocityStop(
  number mInst,
  number axisId)
```

Description: Stop a velocity jog on an the given axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis to stop.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Stops the axis with respect to acceleration settings of the slowest motor that is part of the axis.

Usage:

```
MINSTANCE mInst = 0;
int axis = Z_AXIS;
int rc;
// Jog axis at 10 % max velocity.
rc = mcJogSetRate(mInst, axis, 10);
rc = mcJogVelocityStart(mInst, axis, MC_JOG_POS);
if (rc == MERROR_NOERROR) {
 Sleep(5000);
}
```

```
// Stop the axis.
rc = mcJogVelocityStop(mInst, axis);
```

MPGs

mcMpgGetAccel

C/C++ Syntax:

```
int mcMpgGetAccel(
  MINSTANCE mInst,
  int mpg,
  double *percentMaxAccel);
```

LUA Syntax:

```
percentMaxAccel, rc = mc.mcMpgGetAccel(
  number mInst,
  number mpg)
```

Description: Get the percentage of the maximum acceleration for the given MPG.

Parameters:

Parameter	Description
mInst	The controller instance.
mpg	An integer specifying the MPG.
percentMaxAccel	The address of a double to receive the max acceleration percentage.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**percentMaxAccel** is NULL or **mpg** is out of range.

Notes:
None.

Usage:

```
//
MINSTANCE mInst = 0;
mcMpgGetAccel(MINSTANCE mInst, int mpg, double *percentMaxAccel);
```

mcMpgGetAxis

C/C++ Syntax:

```
int mcMpgGetAxis(
  MINSTANCE mInst,
  int mpg,
  int *axisId);
```

LUA Syntax:

```
axisId, rc = mc.mcMpgGetAxis(
  number mInst,
  number mpg)
```

Description: Retrieve the current axis for the specified MPG.

Parameters:

Parameter	Description
mInst	The controller instance.
mpg	An integer specifying the MPG.
axisId	the address of a integer to revceive the axis ID.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**axisId** is NULL or **mpg** is out of range.

Notes:
None.

Usage:

```
// Get the currently selected axis for MPG 0.
MINSTANCE mInst = 0;
int axisId;
int rc = mcMpgGetAxis(mInst, 0, &axisId;);
```

mcMpgGetCountsPerDetent

C/C++ Syntax:

```
int mcMpgGetCountsPerDetent(
  MINSTANCE mInst,
  int mpg,
  int *pulses);
```

LUA Syntax:

```
pulses, rc = mc.mcMpgGetCountsPerDetent(
  number mInst,
  number mpg)
```

Description: Retrieves the number of pulse required to move 1 unit for the given MPG.

Parameters:

Parameter	Description
mInst	The controller instance.
mpg	An integer specifying the MPG.
pulses	The address of an integer to receive the pulse count.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**pulses** is NULL or **mpg** is out of range.

Notes:
None.

Usage:

```
// Get the counts per detent for MPG 0.
MINSTANCE mInst = 0;
int cnts;
int rc = mcMpgGetCountsPerDetent(mInst, 0, &cnts;);
```

mcMpgGetEncoderReg

C/C++ Syntax:

```
int mcMpgGetEncoderReg(
  MINSTANCE mInst,
  int mpg,
  HMCREG *hReg);
```

LUA Syntax:

```
hReg, rc = mc.mcMpgGetEncoderReg(
  number mInst,
  number mpg)
```

Description: Get the current encoder register that is associated to the given MPG.

Parameters:

Parameter	Description
mInst	The controller instance.
mpg	An integer specifying the MPG.
hReg	The address of a MCHREG to receive the encoder register's handle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**hReg** is NULL or **mpg** is out of range.

Notes:
None.

Usage:

```
// Get the current encoder associated with MPG 0.
MINSTANCE mInst = 0;
HMCREG hReg = 0;

int rc = mcMpgGetEncoderReg(mInst, 0, &hReg;);
```

mcMpgGetInc

C/C++ Syntax:

```
int mcMpgGetInc(
  MINSTANCE mInst,
  int mpg,
  double *inc);
```

LUA Syntax:

```
inc, rc = mc.mcMpgGetInc(
  number mInst,
  number mpg)
```

Description: Retrieve the increment for the given MPG.

Parameters:

Parameter	Description
mInst	The controller instance.
mpg	An integer specifying the MPG.
inc	The address of a double to receive the increment value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**inc** is NULL or **mpg** is out of range.

Notes:
None.

Usage:

```
// Get the incremnt for MPG 0.
MINSTANCE mInst = 0;
double inc;
int rc = mcMpgGetInc(mInst, 0, &inc;);
```

mcMpgGetRate

C/C++ Syntax:

```
int mcMpgGetRate(
  MINSTANCE mInst,
  int mpg,
  double *percentMaxVel);
```

LUA Syntax:

```
percentMaxVel, rc = mc.mcMpgGetRate(
  number mInst,
  number mpg)
```

Description: REtrieve the percentage of the maximum velocity for the given MPG.

Parameters:

Parameter	Description
mInst	The controller instance.
mpg	An integer specifying the MPG.
percentMaxVel	The address of a double to receive the percentage of max velocity.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**rate** is NULL or **mpg** is out of range.

Notes:
None.

Usage:

```
// Get the velocity percentage for MPG 0.
MINSTANCE mInst = 0;
double velPercent;
int rc = mcMpgGetRate(mInst, 0, &velPercent;);
```

mcMpgGetShuttleMode

C/C++ Syntax:

```
int mcMpgGetShuttleMode(
  MINSTANCE mInst,
  BOOL *on);
```

LUA Syntax:

```
on, rc = mc.mcMpgGetShuttleMode(
  number mInst)
```

Description: Determine the state of MPG shuttle mode.

Parameters:

Parameter	Description
mInst	The controller instance.
on	The address of a BOOL to receive the state of MPG shuttle mode.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**on** is NULL.

Notes:
None.

Usage:

```
// Check the state of MPG shuttle mode.
MINSTANCE mInst = 0;
BOOL enabled = FALSE;
int rc = mcMpgGetShuttleMode(mInst, &enabled;);
```

mcMpgMoveCounts

C/C++ Syntax:

```
int mcMpgMoveCounts(
  MINSTANCE mInst,
  int mpg,
  int deltaCounts);
```

LUA Syntax:

```
rc = mc.mcMpgMoveCounts(
  number mInst,
  number mpg,
  number deltaCounts)
```

Description: Move the axis mapped to the given MPG.

Parameters:

Parameter	Description
mInst	The controller instance.
mpg	An integer specifying the MPG.
deltaCounts	An integer specifying the number of counts the MPG has moved.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**mpg** is out of range.

Notes:
This is the main MPG function that a plugin that supports encoders will use. All the plugin has to do is report the delta difference between reads of the encoder.

Usage:

```
// Move the MPG 0 "n" counts.
MINSTANCE mInst = 0;
int n = 16;
int rc = mcMpgMoveCounts(mInst, 0, n);
```

mcMpgSetAccel

C/C++ Syntax:

```
int mcMpgSetAccel(
  MINSTANCE mInst,
  int mpg,
  double percentMaxAccel);
```

LUA Syntax:

```
rc = mc.mcMpgSetAccel(
  number mInst,
  number mpg,
  number percentMaxAccel)
```

Description: Set the acceration percentage for the given MPG.

Parameters:

Parameter	Description
mInst	The controller instance.
mpg	An integer specifying the MPG.
percentMaxAccel	A double specifying the accelration percentage.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**mpg** is out of range.

Notes:
The accelation percentage is based off of the maximum acceleration values for the currently mapped axis.
The percentage is expressed as a whole percentage. e.g. 20.5 for 20 and a half percent.

Usage:

```
// Set the acceleration value for MPG 0.
MINSTANCE mInst = 0;
int rc = mcMpgSetAccel(mInst, 0, 20.5);
```

mcMpgSetAxis

C/C++ Syntax:

```
int mcMpgSetAxis(
  MINSTANCE mInst,
  int mpg,
  int axisId);
```

LUA Syntax:

```
rc = mc.mcMpgSetAxis(
  number mInst,
  number mpg,
  number axis)
```

Description: Set the current driven axis for the given MPG.

Parameters:

Parameter	Description
mInst	The controller instance.
mpg	An integer specifying the MPG.
axisId	An integer specifying the axis.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.
MERROR_INVALID_ARG	**mpg** is out of range.

Notes:
None.

Usage:

```
// Set MPG 0 to move the X axis.
MINSTANCE mInst = 0;
int rc = mcMpgSetAxis(mInst, 0, X_AXIS);
```

mcMpgSetCountsPerDetent

C/C++ Syntax:

```
int mcMpgSetCountsPerDetent(
  MINSTANCE mInst,
  int mpg,
  int pulses);
```

LUA Syntax:

```
rc = mc.mcMpgSetCountsPerDetent(
  number mInst,
  number mpg,
  number pulses);
```

Description: Set the number of counts the MPG outputs for 1 detent. (usually 4)

Parameters:

Parameter	Description
mInst	The controller instance.
mpg	An integer specifying the MPG.
pulses	An integer specifying the number of counts per detent.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**mpg** is out of range.

Notes:
None.

Usage:

```
// Set the number of counts per detent for MPG 0.
MINSTANCE mInst = 0;
int rc = mcMpgSetCountsPerDetent(mInst, 0, 4);
```

mcMpgSetEncoderReg

C/C++ Syntax:

```
int mcMpgSetEncoderReg(
  MINSTANCE mInst,
  int mpg,
  HMCREG hReg);
```

LUA Syntax:

```
rc = mc.mcMpgSetEncoderReg(
  number mInst,
  number mpg
  number hReg)
```

Description: Associate an encoder register to the given MPG.

Parameters:

Parameter	Description
mInst	The controller instance.
mpg	An integer specifying the MPG.
hReg	A MCHREG to associate to the given MPG.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**mpg** is out of range.

Notes:
None.

Usage:

```
// Associate an encoder with MPG 0.
MINSTANCE mInst = 0;
HMCREG hReg = 0;
int rc
rc = mcRegGetHandle(mInst, "/Sim0/Encoder0", &hReg;);
if (rc == MERROR_NOERROR) {
```

```
 rc = mcMpgSetEncoderReg(mInst, 0, hReg);
}
```

mcMpgSetInc

C/C++ Syntax:

```
int mcMpgSetInc(
  MINSTANCE mInst,
  int mpg,
  double inc);
```

LUA Syntax:

```
rc = mc.mcMpgSetInc(
  number mInst,
  number mpg,
  number inc);
```

Description: Set the desired incremnt (.100, .010, .001, etc...) for the given MPG.

Parameters:

Parameter	Description
mInst	The controller instance.
mpg	An integer specifying the MPG.
inc	A double specifying the increment.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**mpg** is out of range.

Notes:
None.

Usage:

```
// Set the increment to .001 for MPG 0.
MINSTANCE mInst = 0;
int rc = mcMpgSetInc(mInst, 0, .001);
```

mcMpgSetRate

C/C++ Syntax:

```
int mcMpgSetRate(
  MINSTANCE mInst,
  int mpg,
  double percentMaxVel);
```

LUA Syntax:

```
rc = mc.mcMpgSetRate(
  number mInst,
  number mpg,
  number percentMaxVel)
```

Description: Set the percentage of the maximum velocity for the given MPG.

Parameters:

Parameter	Description
mInst	The controller instance.
mpg	An integer specifying the MPG.
percentMaxVel	A double specifying the percentage of the maximum velocity.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**mpg** is out of range.

Notes:
The velocity percentage is based off of the maximum velocity value for the currently mapped axis. The percentage is expressed as a whole percentage. e.g. 20.5 for 20 and a half percent.

Usage:

```
// Set the rate for MPG 0 to 25.0%
MINSTANCE mInst = 0;
int rc = mcMpgSetRate(mInst, 0, 25.0);
```

mcMpgSetShuttleMode

C/C++ Syntax:

```
int mcMpgSetShuttleMode(
  MINSTANCE mInst,
  BOOL on);
```

LUA Syntax:

```
rc = mc.mcMpgSetShuttleMode(
  number mInst,
  number on);
```

Description: Set the state of MPG shuttle mode.

Parameters:

Parameter	Description
mInst	The controller instance.
on	A BOOL specifying the state of the MPG shuttle mode.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Turn on MPG shuttle mode.
MINSTANCE mInst = 0;
int rc = mcMpgSetShuttleMode(mInst, MC_ON);
```

Soft Limits

mcSoftLimitGetState

C/C++ Syntax:

```
mcSoftLimitGetState(
 MINSTANCE mInst,
 int axis,
 double *ison);
```

Description: Used to check if the Softlimit for an axis is on or off.

Parameters:

Parameter	Description
mInst	The controller instance.
axis	The axis from which to to get the Softlimit state.
ison	A pointer to a double to receive the status of Softlimit (on or off).

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Can be used to display to the user what the status of the Softlimit for an axis is.

Usage:

```
int mInst = 0;
int axis = Z_AXIS;
dobule IsOn = 0; // DO_OFF to turn off.
mcSoftLimitSetState(mInst, axis, &IsOn;); // Get the softlimit for the Z axis ret
```

mcSoftLimitMaxMinsClear

C/C++ Syntax:

```
mcSoftLimitSetState(
 MINSTANCE mInst,
 int axis,
 int on);
```

Description: Set softlimits to be on or off for an axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axis	The axis to set the softlimit to be on or off.
on	Set the softlimit to be on or off (MC_ON, MC_OFF).

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Can be used at any time. If you enable the softlimits when you are out of bounds any jog will move it back to the softlimit.

Usage:

```
int mInst=0;
int axis = Z_AXIS;
int TurnOn = MC_ON; // MC_OFF to turn off.
mcSoftLimitSetState(mInst, axis, TurnOn); // Set the softlimit forthe Z axis to
```

mcSoftLimitSetState

C/C++ Syntax:

```
mcSoftLimitSetState(
 MINSTANCE mInst,
 int axis,
 int on);
```

Description: Set softlimits to be on or off for an axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axis	The axis to set the softlimit to be on or off.
on	Set the softlimit to be on or off (MC_ON, MC_OFF).

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Can be used at any time. If you enable the softlimits when you are out of bounds any jog will move it back to the softlimit.

Usage:

```
int mInst=0;
int axis = Z_AXIS;
int TurnOn = MC_ON; // MC_OFF to turn off.
mcSoftLimitSetState(mInst, axis, TurnOn); // Set the softlimit forthe Z axis to
```

Spindle

mcSpindleCalcCSSToRPM

C/C++ Syntax:

```
int mcSpindleCalcCSSToRPM(
  MINSTANCE mInst,
  double DiaOfCut,
  BOOL Inch);
```

LUA Syntax:

```
rc = mc.mcSpindleCalcCSSToRPM(
  number mInst,
  number DiaOfCut,
  number Inch);
```

Description: A utility function to implement Constant Surface Speed (CSS) for lathe operation.

Parameters:

Parameter	Description
mInst	The controller instance.
DiaOfCut	A double specifying the diameter of the next cut.
inch	A BOOL specifying if the diameter is in inches (TRUE) or millimiters (FALSE).

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_NOW	The spindle is not in CSS mode.

Notes:
None.

Usage:

```
// make the spindle speed calc the correct RPM for a .550" inch cut.
MINSTANCE mInst = 0;
int rc = mcSpindleCalcCSSToRPM(mInst, .550, TRUE);
```

mcSpindleGetAccelTime

C/C++ Syntax:

```
int mcSpindleGetAccelTime(
  MINSTANCE mInst,
  int Range,
  double *Sec);
```

LUA Syntax:

```
Sec, rc = mc.mcSpindleGetAccelTime(
  number mInst,
  number Range)
```

Description: Retrieve the spindle acceleration time in seconds for the given spindle range.

Parameters:

Parameter	Description
mInst	The controller instance.
Range	An integer specifying the spindle range.
Sec	The address to a double to receive the acceleration time.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_SPIN_RANGE_NOT_FOUND	**Range** specified an invalid spindle range.
MERROR_INVALID_ARG	**Sec** cannot be NULL.

Notes:
None.

Usage:

```
// Get the accel time for spindle range 2.
MINSTANCE mInst = 0;
```

```
double sec = 0;
int rc = mcSpindleGetAccelTime(mInst, 2, &sec;);
```

mcSpindleGetCommandRPM

C/C++ Syntax:

```
int mcSpindleGetCommandRPM(
  MINSTANCE mInst,
  double *RPM);
```

LUA Syntax:

```
RPM, rc = mc.mcSpindleGetCommandRPM(
  number mInst)
```

Description: Retrieve the commanded RPM.

Parameters:

Parameter	Description
mInst	The controller instance.
RPM	The address of a double to receive the commanded RPM.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**RPM** cannot be NULL.

Notes:
None.

Usage:

```
// Get the current commanded RPM
MINSTANCE mInst = 0;
double rpm = 0.0;
int rc = mcSpindleGetCommandRPM(mInst, &rpm;);
```

mcSpindleGetCurrentRange

C/C++ Syntax:

```
int mcSpindleGetCurrentRange(
  MINSTANCE mInst,
  int *Range);
```

LUA Syntax:

```
Range, rc = mc.mcSpindleGetCurrentRange(
  number mInst)
```

Description: Retrieve the current spindle range.

Parameters:

Parameter	Description
mInst	The controller instance.
Range	The address of an integer to receive the current spindle range.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**Range** cannot be NULL.

Notes:
None.

Usage:

```
// Get the current spindle range.
MINSTANCE mInst = 0;
int range = 0;
int rc = mcSpindleGetCurrentRange(mInst, □);
```

mcSpindleGetDecelTime

C/C++ Syntax:

```
int mcSpindleGetDecelTime(
  MINSTANCE mInst,
  int Range,
  double *Sec);
```

LUA Syntax:

```
Sec, rc = mc.mcSpindleGetDecelTime(
  number mInst,
  number Range)
```

Description: Retrieve the decel. time in seconds for the given spindle range.

Parameters:

Parameter	Description
mInst	The controller instance.
Range	An integer specifying the spindle range.
Sec	The address of a double to receive the decel. time.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_SPIN_RANGE_NOT_FOUND	**Range** specified an invalid spindle range.
MERROR_INVALID_ARG	**Sec** cannot be NULL.

Notes:
None.

Usage:

```
// Get the decel. time for spindle range 3
MINSTANCE mInst = 0;
double sec = 0.0;
```

```
int rc = mcSpindleGetDecelTime(mInst, 3, &ec;);
```

mcSpindleGetDirection

C/C++ Syntax:

```
int mcSpindleGetDirection(
  MINSTANCE mInst,
  int *dir);
```

LUA Syntax:

```
dir, rc = mc.mcSpindleGetDirection(
  number mInst)
```

Description: Retrieve the current spindle direction.

Parameters:

Parameter	Description
mInst	The controller instance.
dir	An address to an integer to receive the current spindle direction. (MC_SPINDLE_OFF, MC_SPINDLE_FWD, and MC_SPINDLE_REV)

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**dir** cannot be NULL.

Notes:
None.

Usage:

```
// Get the current spindle direction.
MINSTANCE mInst = 0;
int dir = MC_SPINDLE_OFF;
int rc = mcSpindleGetDirection(mInst, &dir;);
```

mcSpindleGetFeedbackRatio

C/C++ Syntax:

```
int mcSpindleGetFeedbackRatio(
  MINSTANCE mInst,
  int Range,
  double *Ratio);
```

LUA Syntax:

```
Ratio, rc = mc.mcSpindleGetFeedbackRatio(
  number mInst,
  number Range)
```

Description: Retrieve the spindle feedback ratio for the given spindle range.

Parameters:

Parameter	Description
mInst	The controller instance.
Range	An integer specifying the spindle range.
Ratio	The address of a double to receive the feedback ratio.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_SPIN_RANGE_NOT_FOUND	**Range** specified an invalid spindle range.
MERROR_INVALID_ARG	**Ratio** cannot be NULL.

Notes:
None.

Usage:

```
// Get the feedback ratio for spindle range 0.
MINSTANCE mInst = 0;
double ratio = 0.0;
```

```
int rc = mcSpindleGetFeedbackRatio(mInst, 0, :);
```

mcSpindleGetMaxRPM

C/C++ Syntax:

```
int mcSpindleGetMaxRPM(
  MINSTANCE mInst,
  int Range,
  double *MaxRPM);
```

LUA Syntax:

```
MaxRPM, rc = mc.mcSpindleGetMaxRPM(
  number mInst,
  number Range)
```

Description: Retrieve the maximum RPM for the given spindle range.

Parameters:

Parameter	Description
mInst	The controller instance.
Range	An integer specifying the spindle range.
MaxRPM	the address of a double to receive the maximum RPM for the given spindle range.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_SPIN_RANGE_NOT_FOUND	**Range** specified an invalid spindle range.
MERROR_INVALID_ARG	**MaxRPM** cannot be NULL.

Notes:
None.

Usage:

```
// Get the max RPM for range 0.
MINSTANCE mInst = 0;
```

```
double mrpm = 0.0
int rc = mcSpindleGetMaxRPM(mInst, 0, &mrpm;);
```

mcSpindleGetMinRPM

C/C++ Syntax:

```
int mcSpindleGetMinRPM(
  MINSTANCE mInst,
  int Range,
  double *MinRPM);
```

LUA Syntax:

```
MinRPM, rc = mc.mcSpindleGetMinRPM(
  number mInst,
  number Range)
```

Description: Retrieve the minimum RPM for the given spindle range.

Parameters:

Parameter	Description
mInst	The controller instance.
Range	An integer specifying the spindle range.
MaxRPM	the address of a double to receive the minimum RPM for the given spindle range.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_SPIN_RANGE_NOT_FOUND	**Range** specified an invalid spindle range.
MERROR_INVALID_ARG	**MinRPM** cannot be NULL.

Notes:
None.

Usage:

```
// Get the min RPM for range 0.
MINSTANCE mInst = 0;
```

```
double mrpm = 0.0
int rc = mcSpindleGetMinRPM(mInst, 0, &mrpm;);
```

mcSpindleGetMotorAccel

C/C++ Syntax:

```
int mcSpindleGetMotorAccel(
  MINSTANCE mInst,
  int Range,
  double *Accel);
```

LUA Syntax:

```
Accel, rc = mc.mcSpindleGetMotorAccel(
  number mInst,
  number Range)
```

Description: Retrieve the spindle motor acceleration value.

Parameters:

Parameter	Description
mInst	The controller instance.
Range	An integer specifying the spindle range.
Accel	The address to a deouble to receive the spindle acceleration for the given range.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_SPIN_RANGE_NOT_FOUND	**Range** specified an invalid spindle range.
MERROR_INVALID_ARG	**Accel** cannot be NULL.

Notes:
None.

Usage:

```
// Get the spindle acceleration for range 1.
MINSTANCE mInst = 0;
```

```
double accel = 0.0;
int rc = mcSpindleGetMotorAccel(mInst, 1, &accel;);
```

mcSpindleGetMotorMaxRPM

C/C++ Syntax:

```
int mcSpindleGetMotorMaxRPM(
  MINSTANCE mInst,
  double *RPM);
```

LUA Syntax:

```
RPM, rc = mc.mcSpindleGetMotorMaxRPM(
  number mInst)
```

Description: Retrieve the maximum defined RPM for the spindle motor.

Parameters:

Parameter	Description
mInst	The controller instance.
RPM	The address of a double to receive the RPM.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**RPM** cannot be NULL.

Notes:
None.

Usage:

```
// Get the maximum spindle motor RPM.
MINSTANCE mInst = 0;
double rpm = 0.0;
int rc = mcSpindleGetMotorMaxRPM(mInst, &rpm;);
```

mcSpindleGetMotorRPM

C/C++ Syntax:

```
int mcSpindleGetMotorRPM(
  MINSTANCE mInst,
  double *RPM);
```

LUA Syntax:

```
rpm, rc = mc.mcSpindleGetMotorRPM(
  number mInst)
```

Description: Retrieve the current motor RPM.

Parameters:

Parameter	Description
mInst	The controller instance.
RPM	The address to a double to receive the motor RPM.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**RPM** cannot be NULL.

Notes:
This is not the spidnle RPM!!! It is the motor RPM without any ratios for ranges (pulleys), etc...

Usage:

```
// Get the current motor RPM.
MINSTANCE mInst = 0;
double rpm = 0.0;
int rc = mcSpindleGetMotorRPM(mInst, &rpm;);
```

mcSpindleGetOverride

C/C++ Syntax:

```
int mcSpindleGetOverride(
  MINSTANCE mInst,
  double *percent);
```

LUA Syntax:

```
percent, rc = mcSpindleGetOverride(
  number mInst)
```

Description: Retrieve the current spindle override value as a percentage of the commanded RPM.

Parameters: ;

Parameter	Description
mInst	The controller instance.
percent	The address to a double to receive the override decimal percentage.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Get the spindle override. .5 == 50%
MINSTANCE mInst = 0;
double percent = 0.0;
int rc = mcSpindleGetOverride(mInst, &percent;);
```

mcSpindleGetReverse

C/C++ Syntax:

```
int mcSpindleGetReverse(
  MINSTANCE mInst,
  int Range,
  BOOL *Reversed);
```

LUA Syntax:

```
Rreversed, rc = mc.mcSpindleGetReverse(
  number mInst,
  number Range)
```

Description: Retrieve whether the spindle is reversed for the given spindle range.

Parameters:

Parameter	Description
mInst	The controller instance.
Range	An integer specifying the spindle range.
Reversed	The address of a BOOL to receive if the spindle is reversed for the given range.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_SPIN_RANGE_NOT_FOUND	**Range** specified an invalid spindle range.
MERROR_INVALID_ARG	**Reversed** cannot be NULL.

Notes:
None.

Usage:

```
// See if the spindle is reversed in range 1
MINSTANCE mInst = 0;
```

```
BOOL reversed = FALSE;
int mcSpindleGetReverse(mInst, 1, &reversed;);
```

mcSpindleGetSensorRPM

C/C++ Syntax:

```
int mcSpindleGetSensorRPM(
  MINSTANCE mInst,
  double *RPM);
```

LUA Syntax:

```
RPM, rc = mc.mcSpindleGetSensorRPM(
  number mInst)
```

Description: Retrieve the RPM at the spindle sensor.

Parameters:

Parameter	Description
mInst	The controller instance.
RPM	The address of a double to receive the RPM.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**RPM** cannot be NULL.

Notes:
This API function will report RPM of the spindle at the spindle sensor and does not include any feedback ratio.

Usage:

```
// Get the spindle sensor RPM.
MINSTANCE mInst = 0;
double rpm = 0.0;
int rc = mcSpindleGetSensorRPM(mInst, &rpm;);
```

mcSpindleGetSpeedCheckEnable

C/C++ Syntax:

```
int mcSpindleGetSpeedCheckEnable(
  MINSTANCE mInst,
  BOOL *enabled);
```

LUA Syntax:

```
enabled, rc = mc.mcSpindleGetSpeedCheckEnable(
  number mInst)
```

Description: Determine if spindle speed check is enabled.

Parameters:

Parameter	Description
mInst	The controller instance.
enabled	The address of a BOOL that receives the spindle speed check status.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**enabled** cannot be NULL.

Notes:
None.

Usage:

```
// Check the staus of spindle speed check.
MINSTANCE mInst = 0;
BOOL enabled = FALSE;
int rc = mcSpindleGetSpeedCheckEnable(mInst, &enabled;);
```

mcSpindleGetSpeedCheckPercent

C/C++ Syntax:

```
int mcSpindleGetSpeedCheckPercent(
  MINSTANCE mInst,
  double *percent);
```

LUA Syntax:

```
percent, rc = mc.mcSpindleGetSpeedCheckPercent(
  number mInst)
```

Description: Retrieve the spindle speed check percentage.

Parameters:

Parameter	Description
mInst	The controller instance.
percent	The address of a double to receive the speed check percentage.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**percent** cannot be NULL.

Notes:
None.

Usage:

```
// Get the speed check percentage.
MINSTANCE mInst = 0;
double percent = 0.0;
int rc = mcSpindleGetSpeedCheckPercent(mInst, &percent;);
```

mcSpindleGetTrueRPM

C/C++ Syntax:

```
int mcSpindleGetTrueRPM(
  MINSTANCE mInst,
  double *RPM);
```

LUA Syntax:

```
RPM, rc = mc.mcSpindleGetTrueRPM(
  number mInst)
```

Description: Retrieve the true RPM of the spindle, includeing any feedback ratio.

Parameters:

Parameter	Description
mInst	The controller instance.
RPM	The address of a double to receive the true RPM of the spindle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**RPM** cannot be NULL.

Notes:
Calling this API function includes any feedback ratio.

Usage:

```
// Get the true spindle RPM.
MINSTANCE mInst = 0;
double rpm = 0.0;
int rc = mcSpindleGetTrueRPM(mInst, &rpm;);
```

mcSpindleSetAccelTime

C/C++ Syntax:

```
int mcSpindleSetAccelTime(
  MINSTANCE mInst,
  int Range,
  double Sec);
```

LUA Syntax:

```
rc = mc.mcSpindleSetAccelTime(
  number mInst,
  number Range,
  number Sec);
```

Description: Sets the spindle accel. time for the given spindle range.

Parameters:

Parameter	Description
mInst	The controller instance.
Range	An integer specifying the spindle range.
Sec	A double specifying the time in seconds.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_SPIN_RANGE_NOT_FOUND	**Range** specified an invalid spindle range.

Notes:
None.

Usage:

```
// Set the accel. time for spindle range 0.
MINSTANCE mInst = 0;
int rc = mcSpindleSetAccelTime(mInst, 0, 5.0);
```

mcSpindleSetCommandRPM

C/C++ Syntax:

```
int mcSpindleSetCommandRPM(
  MINSTANCE mInst,
  double RPM);
```

LUA Syntax:

```
rc = mc.mcSpindleSetCommandRPM(
  number mInst,
  number RPM)
```

Description: Set the commanded RPM for the spindle.

Parameters:

Parameter	Description
mInst	The controller instance.
RPM	A double specifying the RPM.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Set the spindle RPM to 5000.
MINSTANCE mInst = 0;
int rc = mcSpindleSetCommandRPM(mInst, 5000);
```

mcSpindleSetDecelTime

C/C++ Syntax:

```
int mcSpindleSetDecelTime(
  MINSTANCE mInst,
  int Range,
  double Sec);
```

LUA Syntax:

```
rc = mc.mcSpindleSetDecelTime(
  number mInst,
  number Range,
  number Sec)
```

Description: Set the decel. time for the given spindle range.

Parameters:

Parameter	Description
mInst	The controller instance.
Range	An integer specifying the spindle range.
Sec	A double specifying the time in seconds.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_SPIN_RANGE_NOT_FOUND	**Range** specified an invalid spindle range.

Notes:
None.

Usage:

```
// Set the decel. time for spindle range 0.
MINSTANCE mInst = 0;
in rc = mcSpindleSetDecelTime(mInst, 0, 5);
```

mcSpindleSetDirection

C/C++ Syntax:

```
int mcSpindleSetDirection(
  MINSTANCE mInst,
  int dir);
```

LUA Syntax:

```
rc = mc.mcSpindleSetDirection(
  number mInst,
  number dir)
```

Description: Set the desired spindle direction.

Parameters:

Parameter	Description
mInst	The controller instance.
dir	An integer specifying the spindle direction. (MC_SPINDLE_OFF, MC_SPINDLE_FWD, and MC_SPINDLE_REV)

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_NOW	An attempt was made to set the direction to MC_SPINDLE_FWD or MC_SPINDLE_REV when the control is disabled.
MERROR_INVALID_ARG	**dir** is out of range.

Notes:
None.

Usage:

```
// Set the spindle direction to FWD
MINSTANCE mInst = 0;
```

```
int rc = mcSpindleSetDirection(mInst, MC_SPINDLE_FWD);
```

mcSpindleSetDirectionWait

C/C++ Syntax:

```
int mcSpindleSetDirectionWait(
  MINSTANCE mInst,
  int dir);
```

LUA Syntax:

```
rc = mc.mcSpindleSetDirectionWait(
  number mInst,
  number dir)
```

Description: Set the desired spindle direction and wait for the spindle to come to speed.

Parameters:

Parameter	Description
mInst	The controller instance.
dir	An integer specifying the spindle direction. (MC_SPINDLE_OFF, MC_SPINDLE_FWD, and MC_SPINDLE_REV)

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_NOW	An attempt was made to set the direction to MC_SPINDLE_FWD or MC_SPINDLE_REV when the control is disabled.
MERROR_INVALID_ARG	**dir** is out of range.

Notes:
None.

Usage:

```
// Set the spindle direction to FWD and wait
MINSTANCE mInst = 0;
```

```
int rc = mcSpindleSetDirectionWait(mInst, MC_SPINDLE_FWD);
```

mcSpindleSetFeedbackRatio

C/C++ Syntax:

```
int mcSpindleSetFeedbackRatio(
  MINSTANCE mInst,
  int Range,
  double Ratio);
```

LUA Syntax:

```
rc = mc.mcSpindleSetFeedbackRatio(
  number mInst,
  number Range,
  number Ratio)
```

Description: Set the feedback ratio for the given spindle range.

Parameters:

Parameter	Description
mInst	The controller instance.
Range	An integer specifying the spindle range.
Ratio	A double specifying the feedback ratio.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_SPIN_RANGE_NOT_FOUND	**Range** specified an invalid spindle range.

Notes:
None.

Usage:

```
// Set the feedback ratio for spindle range 0.
MINSTANCE mInst = 0;
int mcSpindleSetFeedbackRatio(mInst, 0, 1.2); // 1.2 : 1
```

mcSpindleSetMaxRPM

C/C++ Syntax:

```
int mcSpindleSetMaxRPM(
  MINSTANCE mInst,
  int Range,
  double RPM);
```

LUA Syntax:

```
rc = mc.mcSpindleSetMaxRPM(
  number mInst,
  number Range,
  number RPM);
```

Description: Set the max RPM for the given spindle range.

Parameters:

Parameter	Description
mInst	The controller instance.
Range	An integer specifying the spindle range.
RPM	A double specifying the maximum RPM.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_SPIN_RANGE_NOT_FOUND	**Range** specified an invalid spindle range.

Notes:
None.

Usage:

```
// Set the max RPM for spindle range 0 to 5000
MINSTANCE mInst = 0;
int mcSpindleSetMaxRPM(mInst, 0, 5000);
```

mcSpindleSetMinRPM

C/C++ Syntax:

```
int mcSpindleSetMinRPM(
  MINSTANCE mInst,
  int Range,
  double RPM);
```

LUA Syntax:

```
rc = mc.mcSpindleSetMinRPM(
  number mInst,
  number Range,
  number RPM);
```

Description: Set the min RPM for the given spindle range.

Parameters:

Parameter	Description
mInst	The controller instance.
Range	An integer specifying the spindle range.
RPM	A double specifying the minimum RPM.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_SPIN_RANGE_NOT_FOUND	**Range** specified an invalid spindle range.

Notes:
None.

Usage:

```
// Set the min RPM for spindle range 0 to 60
MINSTANCE mInst = 0;
int mcSpindleSetMinRPM(mInst, 0, 60);
```

mcSpindleSetMotorAccel

C/C++ Syntax:

```
int mcSpindleSetMotorAccel(
  MINSTANCE mInst,
  int Range,
  double Accel);
```

LUA Syntax:

```
rc = mc.mcSpindleSetMotorAccel(
  MINSTANCE mInst,
  int Range,
  double Accel);
```

Description: Set the spindle motor acceleration for the given spindle range.

Parameters:

Parameter	Description
mInst	The controller instance.
Range	An integer specifying the spindle range.
Accel	A double sepcifying the acceleration value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_SPIN_RANGE_NOT_FOUND	**Range** specified an invalid spindle range.

Notes:
None.

Usage:

```
// Set the acceleration for the spindle motor for range 0
MINSTANCE mInst = 0;
int rc = mcSpindleSetMotorAccel(mInst, 0, 10000);
```

mcSpindleSetMotorMaxRPM

C/C++ Syntax:

```
int mcSpindleSetMotorMaxRPM(
  MINSTANCE mInst,
  double RPM);
```

LUA Syntax:

```
rc = mc.mcSpindleSetMotorMaxRPM(
  number mInst,
  number RPM)
```

Description: Set the maximum spindle motor RPM.

Parameters:

Parameter	Description
mInst	The controller instance.
RPM	A double specifying the maximum spindle motor RPM.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Cap the spindle motor RPM to 6000.
MINSTANCE mInst = 0;
int rc = mcSpindleSetMotorMaxRPM(mInst, 6000);
```

mcSpindleSetOverride

C/C++ Syntax:

```
int mcSpindleSetOverride(
  MINSTANCE mInst,
  double percent);
```

LUA Syntax:

```
rc = mc.mcSpindleSetOverride(
  number mInst,
  number percent)
```

Description: Set the spindle override percentage.

Parameters:

Parameter	Description
mInst	The controller instance.
percent	A double specifying the decimal percentage of spindle override.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Set the spindle override to 50% (.5 == 50%)
MINSTANCE mInst = 0;
int rc = mcSpindleSetOverride(mInst, 0.5);
```

mcSpindleSetRange

C/C++ Syntax:

```
int mcSpindleSetRange(
  MINSTANCE mInst,
  int Range);
```

LUA Syntax:

```
rc = mc.mcSpindleSetRange(
  number mInst,
  number Range)
```

Description: Set the current spindle range.

Parameters:

Parameter	Description
mInst	The controller instance.
Range	An integer specifying the spindle range.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_SPIN_RANGE_NOT_FOUND	**Range** specified an invalid spindle range.

Notes:
None.

Usage:

```
// Set the current spindle range to the first range (0).
MINSTANCE mInst = 0;
mcSpindleSetRange(mInst, 0);
```

mcSpindleSetReverse

C/C++ Syntax:

```
int mcSpindleSetReverse(
  MINSTANCE mInst,
  int Range,
  BOOL Reversed);
```

LUA Syntax:

```
rc = mc.mcSpindleSetReverse(
  number mInst,
  number Range,
  number Reversed)
```

Description: Set the spindle as being reversed for the given spindle range. e.g back gear.

Parameters:

Parameter	Description
mInst	The controller instance.
Range	An integer specifying the spindle range.
Reversed	A BOOL specifying whether the spindle is reversed for the given range.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_SPIN_RANGE_NOT_FOUND	**Range** specified an invalid spindle range.

Notes:
None.

Usage:

```
// Set spindle range 1 as being reversed.
MINSTANCE mInst = 0;
```

```
int rc = mcSpindleSetReverse(mInst, 1, TRUE);
```

mcSpindleSetSensorRPM

C/C++ Syntax:

```
int mcSpindleSetSensorRPM(
  MINSTANCE mInst,
  double RPM);
```

LUA Syntax:

```
rc = mc.mcSpindleSetSensorRPM(
  number mInst,
  number RPM)
```

Description: Set the spindle sensor RPM.

Parameters:

Parameter	Description
mInst	The controller instance.
RPM	A double specifying the spindle sensor RPM.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
This API function is what spindle control plugins should use to report the spindle speed.

Usage:

```
// Set the spindle sensor RPM to 1000
MINSTANCE mInst = 0;
int rc = mcSpindleSetSensorRPM(mInst, 1000);
```

mcSpindleSetSpeedCheckEnable

C/C++ Syntax:

```
int mcSpindleSetSpeedCheckEnable(
  MINSTANCE mInst,
  BOOL enable);
```

LUA Syntax:

```
rc = mc.mcSpindleSetSpeedCheckEnable(
  number mInst,
  number enable)
```

Description: Enable or disable the spindle speed check before feed moves.

Parameters:

Parameter	Description
mInst	The controller instance.
enable	A BOOL specifying the state of spindle speed check.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Spindle speed check is used to make sure the spindle is at an acceptable speed before a feed move.

See also: mcSpindleSetSpeedCheckPercent() and mcSpindleGetSpeedCheckPercent().

Usage:

```
// Enable spindle speed checking.
MINSTANCE mInst = 0;
int rc = mcSpindleSetSpeedCheckEnable(mInst, TRUE);
```

mcSpindleSetSpeedCheckPercent

C/C++ Syntax:

```
int mcSpindleSetSpeedCheckPercent(
  MINSTANCE mInst,
  double percent);
```

LUA Syntax:

```
rc = mc.mcSpindleSetSpeedCheckPercent(
  number mInst,
  number percent)
```

Description: Set the allowable percentage tha the spindle speed can fluctuate.

Parameters:

Parameter	Description
mInst	The controller instance.
percent	A double specifying the percentage.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Set the speed check percentage to 5%.  100 == 100%
MINSTANCE mInst = 0;
int rc = mcSpindleSetSpeedCheckPercent(mInst, 5);
```

mcSpindleSpeedCheck

C/C++ Syntax:

```
int mcSpindleSpeedCheck(
  MINSTANCE mInst,
  int *SpeedOk);
```

LUA Syntax:

```
SpeedOk, rc = mc.mcSpindleSpeedCheck(
  number mInst)
```

Description: Check if the spindle speed is withing the allowable range.

Parameters:

Parameter	Description
mInst	The controller instance.
SpeedOk	The address of a BOOL to receive the spindle speed check status.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_NOW	The spindle is currently off.

Notes:
None.

Usage:

```
// Check to see if the spidnle speed is in the allowed range.
MINSTANCE mInst = 0;
BOOL speedOk;
int rc = mcSpindleSpeedCheck(mInst, &speedOk;);
```

Scripting

mcScriptDebug

C/C++ Syntax:

```
int mcScriptDebug(
  MINSTANCE mInst,
  const char *filename);
```

LUA Syntax:

```
N/A
```

Description: Debug a script.

Parameters: (filename, A string buffer specifying the script file to debug.

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_PLUGIN_NOT_FOUND	There was no registered script plugin for the given script's extension.

Notes:
Script debugging is a function of script plugins. Script plugins register file extensions to identify what script can be used.

Usage:

```
// Debug the M6 macro
MINSTANCE mInst = 0;
int rc = mcScriptDebug(mInst, "m6.mcs");
```

mcScriptEngineRegister

C/C++ Syntax:

```
int mcScriptEngineRegister(
  MINSTANCE mInst,
  HMCPLUG plugid,
  const char *EngineName,
  const char *EngineDesc,
  const char *FileExtensions);
```

LUA Syntax:

```
N/A
```

Description:

Parameters:

Parameter	Description
mInst	The controller instance.
plugid	A HMCPLUG handle specifying the plugin.
EngineName	A string buffer specifying the script engine name.
EngineDesc	A string buffer specifying the script engine description.
FileExtensions	A string buffer specifying the file extenstions that the script engine uses. Each extension is separated with the pipe symbol "|".

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_PLUGIN_NOT_FOUND	**plugid** is invalid.

Notes:
Script plugins register file extensions to identify what script can be used.

Usage:

```
// Register the LUA script engine.
MINSTANCE mInst = 0;
HMCPLUG id; // From mcPluginInit();
int rc = mcScriptEngineRegister(mInst, id, "wxLua", "wxLua script engine", "mcs|
```

mcScriptExecute

C/C++ Syntax:

```
int mcScriptExecute(
  MINSTANCE mInst,
  const char *filename,
  BOOL async);
```

LUA Syntax:

```
rc = mc.mcScriptExecute(
  number mInst,
  string filename,
  number async)
```

Description: Execute a script in the global script context.

Parameters:

Parameter	Description
mInst	The controller instance.
filename	A string buffere specifying the script file to execute.
async	A BOOL specifying whether the script should be executed asynchronously (TRUE == no waiting on completion

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_PLUGIN_NOT_FOUND	No registered script plugins where found for the the specified script's file name extension.
MERROR_NOT_COMPILED	If the script engine compiles the scripts before run-time

Notes:
No error is returned if the specified script file does not exist. If the file does not exist, or there is a

sharing violation preventing the file from being read, this API function simply returns as a NO-OP.

Usage:

```
// Execute the M6 script and wait on it to complete.
MINSTANCE mInst = 0;
rc = mcScriptExecute(mInst, "m6.mcs", FALSE);
```

mcScriptExecuteIfExists

C/C++ Syntax:

```
int mcScriptExecuteIfExists(
  MINSTANCE mInst,
  const char *filename,
  BOOL async);
```

LUA Syntax:

```
rc = mc.mcScriptExecuteIfExists(
  number mInst,
  string filename,
  number async)
```

Description: Execute a script in the global script context if the specified script exists in the filesystem.

Parameters:

Parameter	Description
mInst	The controller instance.
filename	A string buffere specifying the script file to execute.
async	A BOOL specifying whether the script should be executed asynchronously (TRUE == no waiting on completion

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_PLUGIN_NOT_FOUND	No registered script plugins where found for the the specified script's file name extension.
MERROR_NOT_COMPILED	If the script engine compiles the scripts before run-time, this error may be returned which would be an indication of a script syntax error.

Notes:
No error is returned if the specified script file does not exist. If the file does not exist, or there is a sharing violation preventing the file from being read, this API function simply returns as a NO-OP.

Usage:

```
// Execute the M6 script and wait on it to complete.
MINSTANCE mInst = 0;
rc = mcScriptExecute(mInst, "m6.mcs", FALSE);
```

mcScriptExecutePrivate

C/C++ Syntax:

```
int mcScriptExecutePrivate(
  MINSTANCE mInst,
  const char *filename,
  BOOL async);
```

LUA Syntax:

```
rc = mc.mcScriptExecutePrivate(
  number mInst,
  string filename,
  BOOL async)
```

Description: Execute a script in the private script context.

Parameters:

Parameter	Description
mInst	The controller instance.
filename	A string buffere specifying the script file to execute.
async	A BOOL specifying whether the script should be executed asynchronously (TRUE == no waiting on completion

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_PLUGIN_NOT_FOUND	No registered script plugins where found for the the specified script's file name extension.
MERROR_NOT_COMPILED	If the script engine compiles the scripts before run-time, this error may be returned which would be an indication of a script syntax error.

Notes:
No error is returned if the specified script file does not exist. If the file does not exist, or there is a sharing violation preventing the file from being read, this API function simply returns as a NO-OP.

Usage:

```
// Execute myScript.mcs asynchronously in it's own private environment.
MINSTANCE mInst = 0;
int rc = mcScriptExecutePrivate(mInst, "myScript.mcs", TRUE);
```

mcScriptGetExtensions

C/C++ Syntax:

```
int mcScriptGetExtensions(MINSTANCE mInst,
  char *buf,
  long bufsize);
```

LUA Syntax:

```
N/A
```

Description: Retrieves all registered script engine file name extensions.

Parameters:

Parameter	Description
mInst	The controller instance.
buf	A string buffer to receive the script file name extensions.
bufsize	The length of the string buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**buf** is NULL or **bufsize** is <= 0.

Notes:
All returned extensions are delimited by pipe symbols ("|").

Usage:

```
// Retrieve all of the registered script extensions.
MINSTANCE mInst = 0;
mcScriptGetExtensions(MINSTANCE mInst, char *buf, long bufsize);
```

Status Messages

mcCntlGetLastError

C/C++ Syntax:

```
int mcCntlGetLastError(
  MINSTANCE mInst,
  char *buf,
  size_t bufSize);
```

LUA Syntax:

```
buf, rc = mc.mcCntlGetLastError(
  MINSTANCE mInst)
```

Description: Retrieve the last error/status message from the core.

Parameters:

Parameter	Description
mInst	The controller instance.
buf	A string buffer to receive the next error message on the error message stack.
bufSize	The length of the string buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**buf** was NULL or **bufSize** was 0.
MERROR_NODATA	No message was available.

Notes:
Calling this function removes the returned message from the stack.

Usage:

```
int mInst=0;
char error[128];
mcCntlGetLastError(mInst, error, 128); // Receives the last error messge for cont
```

mcCntlGetLastLogMsg

C/C++ Syntax:

```
int mcCntlGetLastLogMsg(
  MINSTANCE mInst,
  char *buf,
  size_t bufSize);
```

LUA Syntax:

```
buf, rc = mc.mcCntlGetLastLogMsg(
  number mInst)
```

Description: Retrieve the last log message from the core.

Parameters:

Parameter	Description
mInst	The controller instance.
buf	A string buffer to receive the next log message on the log message stack.
bufSize	The length of the string buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**buf** was NULL or **bufSize** was 0.
MERROR_NODATA	No log message was available.

Notes:
Calling this function removes the returned message from the log stack.

Usage:

```
int mInst=0;
char msg[128];
mcCntlGetLastLogMsg(mInst, msg, 128); // Receives the last log messge for contro
```

mcCntlLog

C/C++ Syntax:

```
int mcCntlLog(
 MINSTANCE mInst,
 const char *message,
 const char *file,
 int line);
```

LUA Syntax:

```
rc = mc.mcCntlLog(
 number mInst,
 string message,
 string file,
 number line)
```

Description: Log a message to the core log.

Parameters:

Parameter	Description
mInst	The controller instance.
message	A string buffer containing the message to be logged.
file	A string buffer specifying the source file name.
line	An integer specifying the source line number.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

Logging must be enabled with mcCntlSetLogging() in order for the messages to show in the diagnostic log.

If **file** is NULL, no source file or line information is dispalyed in the log for C/C++ environments. If **line** is -1, no source file or line information is dispalyed in the log for scripting environments.

Usage:

```
// Load G code from a string.
MINSTANCE mInst = 0;
BOOL test = TRUE;
int rc;
if (test == TRUE) {
 // File and line info in the log.
 mcCntlLog(mInst, "test == TRUE!", __FILE__, __LINE__);
} else {
 // No file and line info in the log.
 mcCntlLog(mInst, "test == FALSE!", NULL, 0);
}
```

mcCntlSetLastError

C/C++ Syntax:

```
int mcCntlSetLastError(
  MINSTANCE mInst,
  const char *emsg);
```

LUA Syntax:

```
rc = mc.mcCntlSetLastError(
  number mInst,
  string emsg)
```

Description: Sets the last error in the error stack.

Parameters:

Parameter	Description
mInst	The controller instance.
emsg	A string buffer with the error message.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
The error sent may not be seen by the user if the caller of mcGetLastError() is not showing the last errors.

Usage:

```
MINSTANCE mInst = 0;
char *erbuf = "Tool no longer in the spindle";
// Send an error messge for controller 0.
int rc = mcCntlSetLastError(mInst, erbuf);
```

mcCntlSetLogging

C/C++ Syntax:

```
int mcCntlSetLogging(
  MINSTANCE mInst,
  BOOL enable);
```

LUA Syntax:

```
rc = mc.mcCntlSetLogging(
  number mInst,
  number enable)
```

Description: Enable or disable the log facility.

Parameters:

Parameter	Description
mInst	The controller instance.
enable	A BOOL specifying the state of the logging facility.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Enable logging.
MINSTANCE mInst = 0;
int rc = mcCntlSetLogging(mInst, enable);
```

License

- mcCntlCheckLicenseFeature
- mcCntlGetLicenseData
- mcCntlGetLicenseDataLen
- mcCntlGetLicenseModules

mcCntlCheckLicenseFeature

C/C++ Syntax:

```
int mcCntlCheckLicenseFeature(
  MINSTANCE  mInst,
  const char *licFile,
  const char *requirement,
  const char *feature);
```

LUA Syntax:

```
rc = mcCntlCheckLicenseFeature(
  number  mInst,
  string licFile,
  string requirement,
  string feature);
```

Description: Check a license file for a requirement and feature.

Parameters:

Parameter	Description
mInst	The controller instance.
licFile	The license file name.
requirement	A string defining the license requirement.
feature	A string defining the feature to check.

Returns: (

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	The **feature** parameter cannot be NULL.
MERROR_NODATA	The license service count not be initialized.
MERROR_LIC_BAD_ID	The license file PCID does not match the current PCID.

MERROR_LIC_REQUIREMENT_NOT_FOUND	The specified license requirement does not exist in the license file.
MERROR_LIC_FEATURE_NOT_FOUND	The specified feature does not exist in the license file.
MERROR_LIC_EXPIRED	The time limited license has expired.
MERROR_LIC_BAD_KEY	The license encryption key is invalid.

Notes:

If no explicit path is given for the **licFile** parameter (just a file name), the core looks for the license in the Licenses directory of the installation path. Otherwise, the fully qualified path is used.

Usage:

```
// Check the DarwinLic.dat file for feature M4_DARWIN
MINSTANCE mInst = 0;
int rc = mcCntlCheckLicenseFeature(mInst, "DarwinLic.dat", "NF_DARWIN", "M4_DARWI
if (rc != MERROR_NOERROR) {
 // Feature found!!!
 return;
}
```

mcCntlGetLicenseData

C/C++ Syntax:

```
int mcCntlGetLicenseData(
  MINSTANCE mInst,
  int index,
  char *buf,
  long bufSize);
```

LUA Syntax:

```
buf, rc = mc.mcCntlGetLicenseData(
  number mInst,
  number index)
```

Description: Retrieves the license data for the specified index.

Parameters:

Parameter	Description
mInst	The controller instance.
index	The index of the license data to retrieve.
buf	The address of a character buffer that recieves the Gcode line.
bufSize	The size of the receiving character buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**index** is out of range
MERROR_NODATA	There is no license data for the specidied index.

Notes:

Usage:

```
// Get the license data at index 1.
MINSTANCE mInst = 0;
```

```
long len;
char *data;
long dataLen = 0;
int index = 1;
int rc = mcCntlGetLicenseDataLen(mInst, index, &dataLen;);
if (rc == MERROR_NOERROR && dataLen > 0) {
 data = (char *)malloc(dataLen + 1);
 memset(data, 0, dataLen + 1);
 rc = mcCntlGetLicenseData(mInst, index, data, dataLen);
 if (rc == MERROR_NOERROR) {
  // Success!
 }
}
```

mcCntlGetLicenseDataLen

C/C++ Syntax:

```
int mcCntlGetLicenseDataLen(
  MINSTANCE mInst,
  int index,
  long *bufSize);
```

LUA Syntax:

```
bufSize, rc = mc.mcCntlGetLicenseData(
  number mInst,
  number index)
```

Description: Retrieves the license data length for the specified index.

Parameters:

Parameter	Description
mInst	The controller instance.
index	The index of the license data to retrieve.
bufSize	The address of a long to receive the data length..

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**index** is out of range
MERROR_NODATA	There is no license data for the specidied index.

Notes:

Usage:

```
// Get the license data at index 1.
MINSTANCE mInst = 0;
long len;
char *data;
long dataLen = 0;
```

```
int index = 1;
int rc = mcCntlGetLicenseDataLen(mInst, index, &dataLen;);
if (rc == MERROR_NOERROR && dataLen > 0) {
 data = (char *)malloc(dataLen + 1);
 memset(data, 0, dataLen + 1);
 rc = mcCntlGetLicenseData(mInst, index, data, dataLen);
 if (rc == MERROR_NOERROR) {
  // Success!
 }
}
```

mcCntlGetLicenseModules

C/C++ Syntax:

```
int mcCntlGetLicenseModules(
  MINSTANCE mInst,
  unsigned long long *modules)
```

LUA Syntax:

```
N/A
```

Description: Retrieve the 64 bit number comprising the licensed modules.

Parameters:

Parameter	Description
mInst	The controller instance.
modules	The address of a unsigned lon long to receive the licensed modules..

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Retrieve the licensed modules.
MINSTANCE mInst = 0;
unsigned long long modules = 0;
int rc = mcCntlGetLicenseModules(mInst, &modules;);
```

Alphabetical API Refernce

- mcAxisSetPos
- mcAxisSetSoftlimitEnable
- mcAxisSetSoftlimitMax
- mcAxisSetSoftlimitMin
- mcAxisSetSpindle
- mcAxisSetVel
- mcAxisUnmapMotor
- mcAxisUnmapMotors
- mcAxisUnregister
- mcCntlCheckLicenseFeature
- mcCntlCleanup
- mcCntlCloseGCodeFile
- mcCntlCompileScripts
- mcCntlConfigStart
- mcCntlConfigStop
- mcCntlCycleStart
- mcCntlCycleStop
- mcCntlDryRunToLine
- mcCntlEStop
- mcCntlEnable
- mcCntlFeedHold
- mcCntlFeedHoldState
- mcCntlGcodeExecute
- mcCntlGcodeExecuteWait
- mcCntlGcodeInterpGetData
- mcCntlGcodeInterpGetPos
- mcCntlGetBlockDelete
- mcCntlGetBuild
- mcCntlGetComputerID
- mcCntlGetCoolantDelay
- mcCntlGetCwd
- mcCntlGetDiaMode
- mcCntlGetDistToGo
- mcCntlGetFRO
- mcCntlGetGcodeFileName
- mcCntlGetGcodeLine
- mcCntlGetGcodeLineCount
- mcCntlGetGcodeLineNbr
- mcCntlGetLastError
- mcCntlGetLastLogMsg
- mcCntlGetLicenseData
- mcCntlGetLicenseDataLen
- mcCntlGetLicenseModules
- mcCntlGetLocalVar
- mcCntlGetLogging

- mcCntlGetMachDir
- mcCntlGetMistDelay
- mcCntlGetModalGroup
- mcCntlGetMode
- mcCntlGetOffset
- mcCntlGetOptionalStop
- mcCntlGetPoundVar
- mcCntlGetRRO
- mcCntlGetRunTime
- mcCntlGetSingleBlock
- mcCntlGetState
- mcCntlGetStateName
- mcCntlGetStats
- mcCntlGetToolOffset
- mcCntlGetUnitsCurrent
- mcCntlGetUnitsDefault
- mcCntlGetValue
- mcCntlGetVersion
- mcCntlGotoZero
- mcCntlInit
- mcCntlIsInCycle
- mcCntlIsStill
- mcCntlLoadGcodeFile
- mcCntlLoadGcodeString
- mcCntlLog
- mcCntlMachineStateClear
- mcCntlMachineStatePop
- mcCntlMachineStatePush
- mcCntlMdiExecute
- mcCntlProbeFileClose
- mcCntlProbeFileOpen
- mcCntlReset
- mcCntlRewindFile
- mcCntlSetBlockDelete
- mcCntlSetCoolantDelay
- mcCntlSetDiaMode
- mcCntlSetFRO
- mcCntlSetGcodeLineNbr
- mcCntlSetLastError
- mcCntlSetLogging
- mcCntlSetMistDelay
- mcCntlSetMode
- mcCntlSetOptionalStop
- mcCntlSetPoundVar
- mcCntlSetRRO

- mcCntlSetResetCodes
- mcCntlSetSingleBlock
- mcCntlSetStats
- mcCntlSetValue
- mcCntlStartMotionDev
- mcCntlStopMotionDev
- mcCntlToolChangeManual
- mcCntlWaitOnCycleStart
- mcDeviceGetHandle
- mcDeviceGetInfo
- mcDeviceGetInfoStruct
- mcDeviceGetNextHandle
- mcDeviceRegister
- mcFileHoldAquire
- mcFileHoldReason
- mcFileHoldRelease
- mcFixtureLoadFile
- mcFixtureSaveFile
- mcGuiGetWindowHandle
- mcGuiSetCallback
- mcGuiSetFocus
- mcGuiSetWindowHandle
- mcIoGetHandle
- mcIoGetInfoStruct
- mcIoGetNextHandle
- mcIoGetState
- mcIoGetType
- mcIoGetUserData
- mcIoIsEnabled
- mcIoRegister
- mcIoSetDesc
- mcIoSetName
- mcIoSetState
- mcIoSetType
- mcIoSetUserData
- mcIoSyncSignal
- mcIoUnregister
- mcJogGetAccel
- mcJogGetFollowMode
- mcJogGetInc
- mcJogGetRate
- mcJogGetVelocity
- mcJogIncStart
- mcJogIncStop
- mcJogIsJogging

- mcJogSetAccel
- mcJogSetFollowMode
- mcJogSetInc
- mcJogSetRate
- mcJogSetTraceEnable
- mcJogSetType
- mcJogVelocityStart
- mcJogVelocityStop
- mcMotionClearPlanner
- mcMotionCyclePlanner
- mcMotionCyclePlannerEx
- mcMotionGetAbsPos
- mcMotionGetAbsPosFract
- mcMotionGetBacklashAbs
- mcMotionGetBacklashInc
- mcMotionGetIncPos
- mcMotionGetMoveID
- mcMotionGetPos
- mcMotionGetProbeParams
- mcMotionGetRigidTapParams
- mcMotionGetSyncOutput
- mcMotionGetThreadParams
- mcMotionGetThreadingRate
- mcMotionGetVel
- mcMotionSetCycleTime
- mcMotionSetMoveID
- mcMotionSetPos
- mcMotionSetProbeComplete
- mcMotionSetProbePos
- mcMotionSetStill
- mcMotionSetThreadingRate
- mcMotionSetVel
- mcMotionSync
- mcMotionThreadComplete
- mcMotorGetAxis
- mcMotorGetInfoStruct
- mcMotorGetPos
- mcMotorGetVel
- mcMotorIsHomed
- mcMotorIsStill
- mcMotorMapGetDefinition
- mcMotorMapGetLength
- mcMotorMapGetNPoints
- mcMotorMapGetPoint
- mcMotorMapGetPointCount

- mcMotorMapGetStart
- mcMotorMapSetDefinition
- mcMotorMapSetLength
- mcMotorMapSetNPoints
- mcMotorMapSetPoint
- mcMotorMapSetPointCount
- mcMotorMapSetStart
- mcMotorRegister
- mcMotorSetHomePos
- mcMotorSetInfoStruct
- mcMotorUnregister
- mcMpgGetAccel
- mcMpgGetAxis
- mcMpgGetCountsPerDetent
- mcMpgGetEncoderReg
- mcMpgGetInc
- mcMpgGetRate
- mcMpgGetShuttleMode
- mcMpgMoveCounts
- mcMpgSetAccel
- mcMpgSetAxis
- mcMpgSetCountsPerDetent
- mcMpgSetEncoderReg
- mcMpgSetInc
- mcMpgSetRate
- mcMpgSetShuttleMode
- mcPluginConfigure
- mcPluginCoreLoad
- mcPluginCoreUnload
- mcPluginDiagnostic
- mcPluginEnable
- mcPluginGetEnabled
- mcPluginGetInfoStruct
- mcPluginGetLicenseFeature
- mcPluginGetNextHandle
- mcPluginGetValid
- mcPluginInstall
- mcPluginRegister
- mcPluginRemove
- mcProfileDeleteKey
- mcProfileDeleteSection
- mcProfileExists
- mcProfileGetDouble
- mcProfileGetInt
- mcProfileGetName

- mcProfileGetString
- mcProfileSave
- mcProfileWriteDouble
- mcProfileWriteInt
- mcProfileWriteString
- mcRegGetCommand
- mcRegGetHandle
- mcRegGetInfo
- mcRegGetInfoStruct
- mcRegGetNextHandle
- mcRegGetUserData
- mcRegGetValue
- mcRegGetValueLong
- mcRegGetValueString
- mcRegGetValueStringClear
- mcRegRegister
- mcRegSendCommand
- mcRegSetDesc
- mcRegSetName
- mcRegSetResponse
- mcRegSetType
- mcRegSetUserData
- mcRegSetValue
- mcRegSetValueLong
- mcRegSetValueString
- mcRegUnregister
- mcScriptDebug
- mcScriptEngineRegister
- mcScriptExecute
- mcScriptExecuteIfExists
- mcScriptExecutePrivate
- mcScriptGetExtensions
- mcSignalEnable
- mcSignalGetHandle
- mcSignalGetInfo
- mcSignalGetInfoStruct
- mcSignalGetNextHandle
- mcSignalGetState
- mcSignalMap
- mcSignalSetActiveLow
- mcSignalSetState
- mcSignalUnmap
- mcSignalWait
- mcSoftLimitGetState
- mcSoftLimitMaxMinsClear

- mcSoftLimitSetState
- mcSpindleCalcCSSToRPM
- mcSpindleGetAccelTime
- mcSpindleGetCommandRPM
- mcSpindleGetCurrentRange
- mcSpindleGetDecelTime
- mcSpindleGetDirection
- mcSpindleGetFeedbackRatio
- mcSpindleGetMaxRPM
- mcSpindleGetMinRPM
- mcSpindleGetMotorAccel
- mcSpindleGetMotorMaxRPM
- mcSpindleGetMotorRPM
- mcSpindleGetOverride
- mcSpindleGetReverse
- mcSpindleGetSensorRPM
- mcSpindleGetSpeedCheckEnable
- mcSpindleGetSpeedCheckPercent
- mcSpindleGetTrueRPM
- mcSpindleSetAccelTime
- mcSpindleSetCommandRPM
- mcSpindleSetDecelTime
- mcSpindleSetDirection
- mcSpindleSetDirectionWait
- mcSpindleSetFeedbackRatio
- mcSpindleSetMaxRPM
- mcSpindleSetMinRPM
- mcSpindleSetMotorAccel
- mcSpindleSetMotorMaxRPM
- mcSpindleSetOverride
- mcSpindleSetRange
- mcSpindleSetReverse
- mcSpindleSetSensorRPM
- mcSpindleSetSpeedCheckEnable
- mcSpindleSetSpeedCheckPercent
- mcSpindleSpeedCheck
- mcToolGetCurrent
- mcToolGetData
- mcToolGetDesc
- mcToolGetSelected
- mcToolLoadFile
- mcToolPathCreate
- mcToolPathDelete
- mcToolPathGenerate
- mcToolPathGenerateAbort

- mcToolPathGeneratedPercent
- mcToolPathGetAAxisPosition
- mcToolPathGetARotationAxis
- mcToolPathGetAxisColor
- mcToolPathGetBackColor
- mcToolPathGetDrawLimits
- mcToolPathGetExecution
- mcToolPathGetFollowMode
- mcToolPathGetGenerating
- mcToolPathGetLeftMouseDn
- mcToolPathGetLeftMouseUp
- mcToolPathGetPathColor
- mcToolPathIsSignalMouseClicks
- mcToolPathSetAAxisPosition
- mcToolPathSetARotationAxis
- mcToolPathSetAxisColor
- mcToolPathSetBackColor
- mcToolPathSetDrawLimits
- mcToolPathSetFollowMode
- mcToolPathSetPathColor
- mcToolPathSetSignalMouseClicks
- mcToolPathSetView
- mcToolPathSetZoom
- mcToolPathUpdate
- mcToolSaveFile
- mcToolSetCurrent
- mcToolSetData
- mcToolSetDesc

mcAxisDeref

C/C++ Syntax:

```
int mcAxisDeref(
 MINSTANCE mInst,
 int axisId);
```

LUA Syntax:

```
rc = mc.mcAxisDeref(
 number mInst,
 number axisId)
```

Description: Dereference the specified axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	Axis ID to be dereferenced.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.
MERROR_AXIS_NOT_ENABLED	The axis specified by **axis** is not enabled.

Notes:
The axis will be dereferenced allong with all child motors.

Usage:

```
int mInst = 0;
// Set AXIS0 to deref (could have used X_AXIS).
mcAxisDeref(mInst, AXIS0);
```

mcAxisDerefAll

C/C++ Syntax:

```
int mcAxisDerefAll(
 MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcAxisDerefAll(
 number mInst)
```

Description: Dereference all axes.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
All **ENABLED** axes will be dereferenced.

Usage:

```
int mInst = 0;
// Set all axis to deref.
mcAxisDerefAll(mInst);
```

mcAxisEnable

C/C++ Syntax:

```
int mcAxisEnable(
 MINSTANCE mInst,
 int axisId,
 BOOL enabled);
```

LUA Syntax:

```
rc = mc.mcAxisEnable(
 number mInst,
 number axisId,
 number enabled);
```

Description: Enable or disable the specified axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
enabled	Send true to enable or false to disable axis.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Used to enable or disable axis. Disabling an axis will make motion for the axis impossible.

Usage:

```
int mInst = 0;
// Set Y_AXIS to be disabled.
mcAxisEnable(mInst, Y_AXIS, false);
```

mcAxisGetHomeDir

C/C++ Syntax:

```
int mcAxisGetHomeDir(
  MINSTANCE mInst,
  int axisId,
  int *dir);
```

LUA Syntax:

```
dir, rc = mc.mcAxisGetHomeDir(
  number mInst,
  number axisId)
```

Description: Get the hominig direction for the specified axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
dir	Receives the homing direction. A positive value is home POS. A negative value is home NEG.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
The core provides a container for the axis home direction only. It currently does nothing with it. However, a motion plugin may retrieve the stored direction information and use it.

Usage:

```
int mInst=0;
int dir = 0;
// Get Home offset of the X axis.
```

```
mcAxisGetHomeDir(mInst, X_AXIS, &dir;);
```

mcAxisGetHomeInPlace

C/C++ Syntax:

```
int mcAxisGetHomeInPlace(
  MINSTANCE mInst,
  int axisId,
  BOOL *homeInPlace);
```

LUA Syntax:

```
homeInPlace, rc = mc.mcAxisGetHomeInPlace(
  number mInst,
  number axisId)
```

Description: Determine if the Home In Place flage is set.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	Axis ID.
homeInPlace	Receives the home in place flag value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
The core provides a container for the axis home in place flag only. It currently does nothing with it. However, a motion plugin may retrieve the stored flag and implement a means to home the axis and set machine coordinate 0 at its current location.

Usage:

```
// Get the Home In Place flag for the X axis.
MINSTANCE mInst = 0;
BOOL hip = FALSE;
```

```
mcAxisGetHomeInPlace(mInst, X_AXIS, &hip;);
```

mcAxisGetHomeOffset

C/C++ Syntax:

```
int mcAxisGetHomeOffset(
  MINSTANCE mInst,
  int axisId,
  double *offset);
```

LUA Syntax:

```
offset, rc = mc.mcAxisGetHomeOffset(
  number mInst,
  number axisId)
```

Description: Get the home offset for the specified axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
offset	Receives the home offset.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
The core provides a container for the axis home offset only. It currently does nothing with it. However, a motion plugin may retrieve the stored offset and use it.

Usage:

```
int mInst=0;
int offset = 0;
// Get Home offset of the X axis.
mcAxisGetHomeOrder(mInst, X_AXIS, &offset;);
```

mcAxisGetHomeOrder

C/C++ Syntax:

```
int mcAxisGetHomeOrder(
  MINSTANCE mInst,
  int axisId,
  int *order);
```

LUA Syntax:

```
order, rc = mc.mcAxisGetHomeOrder(
  number mInst,
  number axisId)
```

Description: Get the home order for the specified axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
order	Receives the axis home order number.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Get the home order of the axis so all axis can be homed in the order the user would like.
All axes should be read and then homed in the order requested.

Usage:

```
int mInst=0;
double XAxisPos = 0;
int order = 0;
// Get Home order of the X axis.
mcAxisGetHomeOrder(mInst, X_AXIS, &order);
```

mcAxisGetHomeSpeed

C/C++ Syntax:

```
int mcAxisGetHomeSpeed(
  MINSTANCE mInst,
  int axisId,
  double *percent);
```

LUA Syntax:

```
percent, rc = mc.mcAxisGetHomeSpeed(
  number mInst,
  number axisId)
```

Description:

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
percent	Receives the percentage.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:

The core provides a container for the axis home speed percentage only. It currently does nothing with it. However, a motion plugin may retrieve the stored speed percentage and use it.

Usage:

```
int mInst=0;
int rc = 0;
double XAxisPos = 0;
double percent = 0;
double maxVel = 0
```

```
double homeVel = 0;

// Get home speed percentage of the X axis.
mcAxisGetHomeSpeed(mInst, X_AXIS, &percent;);
mcAxisGetVel(mInst, X_AXIS, &maxVel;);
homeVel = maxVel * percent;
```

mcAxisGetInfoStruct

C/C++ Syntax:

```
int mcAxisGetInfoStruct(
  MINSTANCE mInst,
  int axisId,
  axisinfo_t *ainf);
```

LUA Syntax:

```
N/A
```

Description: Retrieve axis parameters in one call.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	Axis number to be dereffed.
ainf	The address of a axisinfo struct that receives the info for the axis.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
mcAxisGetInfoStruct() is used for quickly retrieving all of the axis parameters. The information is placed in the axisinfo structure. This information can then be modified to update the axis parameters with mcAxisSetInfoStruct().

```
struct axisinfo {
 BOOL OutOfBandAxis;     // Is this an out of band axis?
 BOOL IsStill;           // Set high when the axis is not moving
 BOOL  Jogging;          // Used to tell to jog...
 BOOL  Homing;           // Used to tell the state of the home operation.
 int  Id;                // Axis Id
```

```
 BOOL IsSpindle;          // Does this axis control a spindle?
 BOOL Enabled;            // Is axis enabled?
 BOOL SoftlimitEnabled;   // Softlimits enabled?
 double SoftMaxLimit;     // Count for the max travel.
 double SoftMinLimit;     // Count for the m.
 BOOL VelocityMode;       // Used to make the axis move at a fixed speed
 BOOL BufferJog;          // Buffer all jogs?
 double Pos;              // Position in user units.
 double Mpos;             // Machine position in user units.
 int HomeOrder;           // The order in which to home the axis.
 int HomeDir;             // The direction the axis homes.
 double HomeOffset;       // The offset from the the limits switch.
 double HomeSpeedPercent;// The percentage of the max velocity at which to home.
 BOOL SoftlimitUsed;      // Use Softlimits?
 BOOL HomeInPlace;        // Zero the axis in place when Refed?
 int MotorId[MC_MAX_AXIS_MOTORS]; //child motor ID array.
};
typedef struct axisinfo axisinfo_t;
```

Usage:

```
int mInst = 0;
axisinfo_t ainf;
// Get Y_AXIS info structure.
mcAxisGetInfoStruct(mInst, Y_AXIS, &ainf;);
```

mcAxisGetMachinePos

C/C++ Syntax:

```
int mcAxisGetMachinePos(
 MINSTANCE mInst,
 int axisId,
 double *val);
```

LUA Syntax:

```
val, rc = mc.mcAxisGetMachinePos(
 number mInst,
 number axisId)
```

Description: Get the machine positon of an axis in User units.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
val	Receives the axis machine position in User units.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Get the position of the axis in User units relitive to the home position.

Usage:

```
int mInst=0;
double XAxisMachinePos = 0;
int AxisNumber = X_AXIS;
mcAxisGetMachinePos(mInst, AxisNumber, &XAxisMachinePos;); // Get the position o
```

mcAxisGetMotorId

C/C++ Syntax:

```
int mcAxisGetMotorId(
  MINSTANCE mInst,
  int axis,
  int childId,
  int *motorId);
```

LUA Syntax:

```
motorId, rc = mc.mcAxisGetMotorId(
  number mInst,
  number axis,
  number childId)
```

Description: Get the motor IDs of the motors used on the axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axis	The axis ID from which to get the position.
childId	The axis ID from which to get the position.
motorId	Receives the motor id of the child motor.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.
The motor specified in **child_id** was not found.	

Notes:
Get the motor ID of the child motor. The axis can have many child motors so to get all motors search until MERROR_MOTOR_NOT_FOUND The motor specified by **motorId** was not found. or return is no longer MERROR_NOERROR No Error.

Usage:

```
int mInst=0;
int motorIds[]={-1,-1,-1,-1,-1}
int id;
// Get all the motor ID's for the X axis up to 5 motors.
for(int i = 0; i < 5; i++){
 if (mcAxisGetMotorId(mInst, X_AXIS, i, &id;) ==

MERROR_NOERROR No Error.) {
  motorIds[i] = id;
 }
}
```

mcAxisGetOverrideAxis

C/C++ Syntax:

```
int mcAxisGetOverrideAxis(
  MINSTANCE mInst,
  int axis,
  int *axis1,
  int *axis2,
  int *axis3,
  int *axis4);
```

LUA Syntax:

```
axis1, axis2, axis3, axis4, rc = mc.mcAxisGetOverrideAxis(
  number mInst,
  number axis)
```

Description: Get axis override axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axis	The axis id to apply override axis too.
axis1	Receives the override axis1 ID.
axis2	Receives the override axis2 ID.
axis3	Receives the override axis3 ID.
axis4	Receives the override axis4 ID.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Get the override axis ID for each override axis on the main axis. override_axis_id will return 0 if unused.

Usage:

```
int mInst = 0;
int ora[4];
// Get the override axis for the Z axis.
mcAxisGetOverrideAxis(mInst, ZAXIS, &ora;[0], &ora;[1], &ora;[2], &ora;[3]);
```

mcAxisGetPos

C/C++ Syntax:

```
int mcAxisGetPos(
 MINSTANCE mInst,
 int axisId,
 double *val);
```

LUA Syntax:

```
val, rc = mc.mcAxisGetPos(
 number mInst,
 number axisId)
```

Description: Get the positon of an axis in user units.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID from which to get the position.
val	Receives the axis position in User units.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
The axis position reported is in user units relative to the fixture and tool offsets that are active. This would be the typical value that the user would see in an axis DRO on the GUI of the controller.

Usage:

```
int mInst=0;
double XAxisPos = 0;
int AxisNumber = X_AXIS;
// Get the position of the X axis.
mcAxisGetPos(mInst, AxisNumber, &XAxisPos;);
```

mcAxisGetProbePos

C/C++ Syntax:

```
int mcAxisGetProbePos(
  MINSTANCE mInst,
  int axisId,
  BOOL machinePos,
  double *val);
```

LUA Syntax:

```
val, rc = mc.mcAxisGetProbePos(
  number mInst,
  number axisId,
  number machinePos)
```

Description: Retrieves the position of the last probe strike for the given axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	Axis ID.
machinePos	TRUE for machine coordinates

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.
MERROR_INVALID_ARG	The **val** pointer cannot be NULL.

Notes:

Usage:

```
// Get the last probe strike position for the X axis
MINSTANCE mInst = 0;
double xProbePos = 0;
```

```
mcAxisGetProbePos(mInst, X_AXIS, FALSE, &xProbPos;);
```

mcAxisGetProbePosAll

C/C++ Syntax:

```
int mcAxisGetProbePosAll(
  MINSTANCE mInst,
  BOOL machinePos,
  double *x,
  double *y,
  double *z,
  double *a,
  double *b,
  double *c);
```

LUA Syntax:

```
x, y, z, a, b, c, rc = mc.mcAxisGetProbePosAll(
  number mInst,
  number machinePos)
```

Description: Retrieves the position of the last probe strike for all coordinated axes.

Parameters:

Parameter	Description
mInst	The controller instance.
machinePos	TRUE for machine coordinates
x	The X position.
y	The Y position.
z	The Z position.
a	The A position.
b	The B position.
c	The C position.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

NULL can be specified for any of the parameters x, y, z, a, b, or c.

Usage:

```
// Get the last probe strike position for the X, Y, and Z axes.
MINSTANCE mInst = 0;
double x, y, z;
int rc = mcAxisGetProbePosAll(mInst, FASLE, &x;, &y;, &z;, NULL, NULL, NULL);
if (rc == MERROR_NOERROR) {
 // Process probe positions...
}
```

mcAxisGetScale

C/C++ Syntax:

```
int mcAxisGetScale(
  MINSTANCE mInst,
  int axisId,
  double *scaleVal);
```

LUA Syntax:

```
scaleVal, rc = mc.mcAxisGetProbePos(
  number mInst,
  number axisId)
```

Description: Retrieves the current scaling value for the given axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	Axis ID.
scaleVal	Receives the scale value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.
MERROR_INVALID_ARG	The **scaleVal** pointer cannot be NULL

Notes:
Scale values that are negative produce mirror images.

Usage:

```
// Get the scale value for the X axis
MINSTANCE mInst = 0;
double scale = 0;
mcAxisGetScale(mInst, X_AXIS, &scale;);
```

mcAxisGetSoftlimitEnable

C/C++ Syntax:

```
int mcAxisGetSoftlimitEnable(
   MINSTANCE mInst,
   int axisId,
   int *enable);
```

LUA Syntax:

```
enable, rc = mc.mcAxisGetSoftlimitEnable(
   number mInst,
   number axisId)
```

Description: Get the soft limit enable status.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
enable	Receives the softlimit enable status.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Soft limits can be enabled on a per axis basis. This is the master soft limit enable for the axis. It overrides the soft limit enable set by mcSoftLimitSetState().

Usage:

```
// Get master soft limit enable state for axis 0.
int mInst = 0;
int enable = 0;
mcAxisGetSoftlimitEnable(mInst, 0, &enable;);
```

mcAxisGetSoftlimitMax

C/C++ Syntax:

```
int mcAxisGetSoftlimitMax(
  MINSTANCE mInst,
  int axisId,
  double *max);
```

LUA Syntax:

```
max, rc = mc.mcAxisGetSoftlimitMin(
  number mInst,
  number axisId)
```

Description: Get the softlimit maximum for the axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
max	Receives the maximum poisition of the axis in machine position.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Get the maximum position of the axis based on the least common denominator of all the child motors used on the axis.

Usage:

```
int mInst=0;
int axis = Y_AXIS;
double min = 0;
```

```
mcAxisGetSoftlimitMax(mInst, axis, &min;);// Get the max distance of the softlim
```

mcAxisGetSoftlimitMin

C/C++ Syntax:

```
int mcAxisGetSoftlimitMin(
  MINSTANCE mInst,
  int axisId,
  double *min);
```

LUA Syntax:

```
min, rc = mc.mcAxisGetSoftlimitMin(
  number mInst,
  number axisId)
```

Description: Get the softlimit minimum for the axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
min	Receives the minimum poisition of the axis in machine position.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Get the minimum position of the axis based on the least common denominator of all the child motors used on the axis.

Usage:

```
int mInst=0;
int axis = Y_AXIS;
double min = 0;
```

```
mcAxisGetSoftlimitMin(mInst, axis, &min;);// Get the min distance of the softlim
```

mcAxisGetSpindle

C/C++ Syntax:

```
int mcAxisGetSpindle(
  MINSTANCE mInst,
  int axisId,
  bool *spindle);
```

LUA Syntax:

```
spindle, rc = mcAxisGetSpindle(
  number mInst,
  number axisId)
```

Description: Determine if the specified axis is an axis that controls a spindle.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
spindle	Receives the spindle flag value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Only an out of band axis can control a spindle.

Usage:

```
// Find the spindle axis
int mInst = 0;
int axisId;
bool isSpindle = false;
for (axisId = MC_MAX_COORD_AXES; axisId < MC_MAX_AXES; axisId++) {
    mcAxisGetSpindle(mInst, axisId, &isSpindle;);
```

```
    if (isSpindle) {
        // Spindle found!
        break;
    }
}
```

mcAxisGetVel

C/C++ Syntax:

```
int mcAxisGetVel(
 MINSTANCE mInst,
 int axisId,
 double *velocity);
```

LUA Syntax:

```
velocity, rc = mc.mcAxisGetVel(
 number mInst,
 number axisId)
```

Description: Get the velocity of an axis in user units per min.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
velocity	Receives the velocity of the axis in user units per min.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
To get the blended velocity sum the velocity vectors:
blendVel = sqrt(vX*vX + vY*vY + vZ*vZ)

Usage:

```
int mInst=0;
int axis = Y_AXIS;
double CurrentVel = 0;
mcAxisGetVel(mInst, axis, &CurrentVel;); // Get the current speed of the Y axis.
```

mcAxisHome

C/C++ Syntax:

```
int mcAxisHome(
 MINSTANCE mInst,
 int axisId);
```

LUA Syntax:

```
rc = mc.mcAxisHome(
 number mInst,
 number axisId)
```

Description: Used to start an axis homing.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID of which to set the position.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.
MERROR_NOT_NOW	The operation could not be completed at this time.

Notes:
Home the axis, all motors will be homed that are child of the axis.

Usage:

```
int mInst=0;
int AxisNumber = X_AXIS;
// Set the X axis to home.
mcAxisSetPos(mInst, AxisNumber);
```

mcAxisHomeAll

C/C++ Syntax:

```
int mcAxisHomeAll(
 MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcAxisHomeAll(
 number mInst)
```

Description: Used to start an axis homing.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_NOW	The operation could not be completed at this time.

Notes:
Home the all the axes. Only the coordinated axes will be homed.

Usage:

```
int mInst=0;
// Home all coordinated axes.
mcAxisHomeAll(mInst);
```

mcAxisHomeComplete

C/C++ Syntax:

```
int mcAxisHomeComplete(
  MINSTANCE mInst,
  int axisId);
```

LUA Syntax:

```
rc = mc.mcAxisHomeComplete(
  number mInst,
  number axisId);
```

Description: Report that the axis is finished homing.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID which is finished homing.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Reports that the axis is finished homming. This is done by the motion plugin to report when it is finished the homing operation. The motors should be at their home positions and marked as still before this call is made.

Usage:

```
int mInst=0;
int AxisNumber = X_AXIS;
// Mark the X axis home operation as complete.
mcAxisHomeComplete(mInst, AxisNumber);
```

mcAxisHomeCompleteWithStatus

C/C++ Syntax:

```
int mcAxisHomeCompleteWithStatus(
  MINSTANCE mInst,
  int axisId
  BOOL success);
```

LUA Syntax:

```
rc = mc.mcAxisHomeCompleteWithStatus(
  number mInst,
  number axisId
  number success);
```

Description: Report that the axis is finished homing and provide the status of the home operation.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID which is finished homing.
success	The status of the home operation. TRUE or FALSE

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Reports that the axis is finished homming. This is done by the motion plugin to report when it is finished. The motors should be at their home positions and marked as still before this call is made. If the success is set to FALSE, the axis is dereferenced (not homed).

Usage:

```
int mInst=0;
int AxisNumber = X_AXIS;
```

```
// Mark the X axis homing operation as finished and that it completed with succes
mcAxisHomeCompleteWithStatus(mInst, AxisNumber, TRUE);
```

mcAxisIsEnabled

C/C++ Syntax:

```
int mcAxisIsEnabled(
  MINSTANCE mInst,
  int axisId,
  BOOL *enabled);
```

LUA Syntax:

```
enabled, rc = mc.mcAxisIsEnabled(
  number mInst,
  number axisId)
```

Description: Determines if the specified axis is enabled.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
enabled	Receives the axis enable flag value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
This function is primarily used to determine if the user has enabled the axis in the control configuration. It is **NOT** the state of the motor enable signal (OSIG_ENABLEx).

Usage:

```
// Find all enabled axes.
int mInst = 0;
int axisId;
BOOL enabled = FALSE;
for (axisId = 0; axisId < MC_MAX_AXES; axisId++) {
```

```
    enabled = false;
    mcAxisIsEnabled(mInst, axisId, &enabled;);
    if (enabled == TRUE) {
        // Axis is enabled!
    }
}
```

mcAxisIsHomed

C/C++ Syntax:

```
int mcAxisIsHomed(
 MINSTANCE mInst,
 int axisId,
 BOOL *homed);
```

LUA Syntax:

```
homed, rc= mc.mcAxisIsHomed(
  number mInst,
  number axisId)
```

Description: Check to see if axis has been homed.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
homed	Receives the axis homed state (TRUE homed, FALSE not homed).

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Get the homed state of the axis. Axis will only report back as homed if all child motors have been homed.

Usage:

```
int mInst = 0;
BOOL homed = FALSE;
// Get the homed state of Z axis.
```

```
mcAxisIsHomed(mInst, Z_AXIS, &homed;);
if (rc == MERROR_NOERROR && homed == TRUE) {
 // Z is homed.
}
```

mcAxisIsHoming

C/C++ Syntax:

```
int mcAxisIsHomin(
 MINSTANCE mInst,
 int axisId,
 BOOL *homing);
```

LUA Syntax:

```
homing, rc= mc.mcAxisIsHomed(
  number mInst,
  number axisId)
```

Description: Check to see if an axis is in a home operation.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
homing	Receives the axis himing state (TRUE homing, FALSE not homing).

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:

Usage:

```
int mInst = 0;
BOOL homing = FALSE;
// Get the homing state of Z axis.
mcAxisIsHoming(mInst, Z_AXIS, &homing;);
if (rc == MERROR_NOERROR && homing == TRUE) {
 // Z is in a home operation.
```

}

mcAxisIsStill

C/C++ Syntax:

```
int mcAxisIsStill(
 MINSTANCE mInst,
 int axisId,
 BOOL *still);
```

LUA Syntax:

```
still, rc = mc.mcAxisIsStill(
  number mInst,
  number axisId)
```

Description: Report if the axis is still.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
still	Receives a BOOL reflecting the still state.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:

Reports back TRUE if the motor is still and FALSE if it is moving. The data is pulled from the axis' child motors. All of the motors that are mapped to the axis must be still in order for the axis to be marked as still.

Note that an axis can be marked as still even when G code is still processing. A report that the axis is still simply means that the axis is not moving at the point in time the function is called.

Usage:

```
int mInst=0;
int AxisNumber = X_AXIS;
int still = 0;
// Get the is moving state of the axis vlue of 1 is stopped.
mcAxisIsStill(mInst, AxisNumber, &still;);
```

mcAxisMapMotor

C/C++ Syntax:

```
int mcAxisMapMotor(
  MINSTANCE mInst,
  int axisId,
  int motorId);
```

LUA Syntax:

```
rc = mc.mcAxisMapMotor(
  number mInst,
  number axisId,
  number motorId)
```

Description: Map a motor to an axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The ID of the axis to which to map the motor.
motorId	The ID of the motor to map to the specified axis.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
Set the motor to be a child of the axis. Up to six motors can be mapped to an axis. The fist mapped motor is the master motor. Any additional mapped motors are slave motors. Softlimits will be the least common denominator for all the mapped motors.

Usage:

```
int mInst=0;
int AxisNumber = X_AXIS;
int MotorNumber = MOTOR5
// Map motor5 to the X axis.
mcAxisMapMotor(mInst, AxisNumber, MotorNumber);
```

mcAxisRegister

C/C++ Syntax:

```
int mcAxisRegister(
  MINSTANCE mInst,
  int axisId);
```

LUA Syntax:

```
rc = mc.mcAxisRegister(
  number mInst,
  number axisId);
```

Description: Register an axis to the system.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_CREATED	The axis was not created.

Notes:

Register an axis to the system so motors can then be mapped to it. This could be a coordinated axis or an out of band axis.

This function is not used as the core registers axes based on capabilities set by the runtime license.

Usage:

```
int mInst=0;
for (int a = AXIS0; < AXIS4; a++) {
    // Register axis0 - axis2 to the core.
    if( mcAxisRegister(mInst, a) !=
```

```
MERROR_NOERROR No Error.){
          return(false);
     }
}
```

mcAxisRemoveOverrideAxis

C/C++ Syntax:

```
int mcAxisRemoveOverrideAxis(
  MINSTANCE mInst,
  int axisId,
  int overrideId);
```

LUA Syntax:

```
rc = mcAxisRemoveOverrideAxis(
  number mInst,
  number axisId,
  number overrideId)
```

Description: Set axis as an override axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
overrideId	The axis ID of the override axis to be removed.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Remove override axis from the main axis.

Usage:

```
int mInst = 0;
int homed = 0;
// Remove axis 8 from the Z axis.
mcAxisRemoveOverrideAxis( mInst, ZAXIS, AXIS_8);
```

mcAxisSetHomeDir

C/C++ Syntax:

```
int mcAxisSetHomeDir(
  MINSTANCE mInst,
  int axisId,
  int dir);
```

LUA Syntax:

```
rc = mc.mcAxisSetHomeDir(
  number mInst,
  number axisId,
  number dir)
```

Description: Set the specified axis' homing direction.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
dir	The homing direction. -1 (NEG) or 1 (POS)

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
The core provides a container for the axis home direction only. It currently does nothing with it. However, a motion plugin may retrieve the stored direction information and use it.

Usage:

```
// Set axis 0 homing directio to POS.
int mInst = 0;
mcAxisSetHomeDir(mInst, 0, 1);
```

mcAxisSetHomeInPlace

C/C++ Syntax:

```
int mcAxisSetHomeInPlace(
  MINSTANCE mInst,
  int axisId,
  BOOL homeInPlace);
```

LUA Syntax:

```
rc = mc.mcAxisGetHomeInPlace(
  number mInst,
  number axisId,
  number homeInPlace)
```

Description: Set the axis' Home In Place flag.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	Axis ID.
homeInPlace	TRUE or FALSE

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
The core provides a container for the axis home in place flag only. It currently does nothing with it. However, a motion plugin may retrieve the stored flag and implement a means to home the axis and set machine coordinate 0 at its current location.

Usage:

```
// Set the Home In Place flag for the X axis.
MINSTANCE mInst = 0;
```

```
BOOL hip = FALSE;
mcAxisGetHomeInPlace(mInst, X_AXIS, hip);
```

mcAxisSetHomeOffset

C/C++ Syntax:

```
int mcAxisSetHomeOffset(
  MINSTANCE mInst,
  int axisId,
  double offset);
```

LUA Syntax:

```
rc = mcAxisSetHomeOffset(
  number mInst,
  number axisId,
  number offset);
```

Description:

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
offset	Receives the axis home offset..

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:

The core provides a container for the axis home offset only. It currently does nothing with it. However, a motion plugin may retrieve the stored offset and use it.

Usage:

```
// Set the home offset for axis 0.
int mInst = 0;
mcAxisSetHomeOffset(mInst, 0, 1.5 /* inches */);
```

mcAxisSetHomeOrder

C/C++ Syntax:

```
int mcAxisSetHomeOrder(
  MINSTANCE mInst,
  int axisId,
  int order);
```

LUA Syntax:

```
rc = mc.mcAxisSetHomeOrder(
  number mInst,
  number axisId,
  number order)
```

Description: Set the axis home order.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
order	The order in which the axis will be homed (0 is first).

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Set the order the axis is homed in. 0 is the first axis to get homed and more then one axis could have the same home order number.

Usage:

```
int mInst=0;
// Set the order of homming so the Z axis will home first then the X and Y.
mcAxisSetHomeOrder(mInst, Z_AXIS , 0);
mcAxisSetHomeOrder(mInst, X_AXIS , 1);
```

```
mcAxisSetHomeOrder(mInst, Y_AXIS , 1);
```

mcAxisSetHomeSpeed

C/C++ Syntax:

```
int mcAxisSetHomeSpeed(
  MINSTANCE mInst,
  int axis,
  double percent);
```

LUA Syntax:

```
rc = mc.mcAxisSetHomeSpeed(
  number mInst,
  number axis,
  number percent);
```

Description: Set the axis homing velocity as a percentage of the axis maximum velocity.

Parameters:

Parameter	Description
mInst	The controller instance.
axis	The axis ID.
percent	The percentage of the axis maximum velocity expressed in decimal. e.g. 20.5 for 20.5%

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
The core provides a container for the axis home speed percentage only. It currently does nothing with it. However, a motion plugin may retrieve the stored speed percentage and use it.

Usage:

```
// Set the homing speed for axis 0 as a percentage of the max velocity.
int mInst = 0;
```

```
mcAxisSetHomeSpeed(mInst, 0, 20.5 /* percent */);
```

mcAxisSetInfoStruct

C/C++ Syntax:

```
int mcAxisSetInfoStruct(
  MINSTANCE mInst,
  int axisID,
  axisinfo_t *ainf);
```

LUA Syntax:

```
N/A
```

Description: Get axis info struct to get all axis settings in one call.

Parameters:

Parameter	Description
mInst	The controller instance.
axisID	The axis ID.
ainf	The address of a axisinfo_t sruct that receives the info for the axis.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Set all the Axis information in the axis information structure in one call. This is used when all the data needs to be set. Call is best used when paired with mcAxisGetInfoStruct().

```
struct axisinfo {
 bool OutOfBandAxis;
 bool IsStill;           // Set high when the axis is not moving
 int  Jogging;           // Used to tell to jog...
 int  Homing;            // Used to tell the state of the home operation.
 int  Id;                // Axis Id
 bool IsSpindle;   // Does this axis control a spindle?
 bool Enabled;           // Is axis enabled?
```

```
 bool SoftlimitEnabled;  // Softlimits enabled?
 double SoftMaxLimit;    // Count for the max travel.
 double SoftMinLimit;    // Count for the min travel.
 bool VelocityMode;      // Used to make the axis move at a fixed speed
 bool BufferJog;
 double Pos;             // Position in user units.
 double Mpos;            // Machine position in user units.
 int HomeOrder;          // The order in which to home the axis.
};
```

Usage:

```
int mInst = 0;
axisinfo_t ainf;

mcAxisGetInfoStruct(mInst, Y_AXIS, &ainf;);// Get Y_AXIS info structure.
ainf.HomeOrder = 1;// Set Y_AXIS home order to 1.
mcAxisSetInfoStruct(mInst, Y_AXIS, &ainf;);// Set Y_AXIS info structure.
```

mcAxisSetMachinePos

C/C++ Syntax:

```
int mcAxisSetMachinePos(
  MINSTANCE mInst,
  int axis,
  double val);
```

LUA Syntax:

```
rc = mc.mcAxisSetMachinePos(
  number mInst,
  number axis,
  number val)
```

Description:

Parameters:

Parameter	Description
mInst	The controller instance.
axis	The axis ID.
val	The desired axis position to apply a fixture offset.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:

Usage:

```
// Set axis 0 fisxture offset.
int mInst = 0;
mcAxisSetMachinePos(mInst, 0, 0.0 /* set part zero */);
```

mcAxisSetOverrideAxis

C/C++ Syntax:

```
int mcAxisSetOverrideAxis(
  MINSTANCE mInst,
  int axis,
  int overrideId);
```

LUA Syntax:

```
rc = mc.mcAxisSetOverrideAxis(
  number mInst,
  number axis,
  number overrideId)
```

Description: Set axis as an override axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axis	The axis ID to apply override axis too.
overrideId	The axis ID of the of the axis to be used as an override.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Setting an override axis allows an axis to be superimposed on top of the main axis. This can be used as an axis to adjust the main axis on the fly (like Torch Height Control). Up to 4 override axes can be used at the same time.

Usage:

```
int mInst = 0;
int homed = 0;
```

```
// Set the axis 8 to be the override axis for the Z axis.
mcAxisSetOverrideAxis(mInst, ZAXIS, AXIS_8);
```

mcAxisSetPos

C/C++ Syntax:

```
int mcAxisSetPos(
 MINSTANCE mInst,
 int axisId,
 double val);
```

LUA Syntax:

```
rc = mc.mcAxisSetPos(
 number mInst,
 number axisId,
 number val);
```

Description: Set the Position of an axis by changing the fixture offset.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID for which to set the position.
val	The axis position (in user units).

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Set the position of the axis by changing the current Fixture offset. This is not valid for out of band axes.

Usage:

```
int mInst=0;
double XAxisPos = .5;
int AxisNumber = X_AXIS;
// Set the position of the X axis by changing the fixture offset.
```

```
mcAxisSetPos(mInst, AxisNumber, XAxisPos);
```

mcAxisSetSoftlimitEnable

C/C++ Syntax:

```
int mcAxisSetSoftlimitEnable(
   MINSTANCE mInst,
   int axis,
   int enabel);
```

LUA Syntax:

```
rc = mc.mcAxisSetSoftlimitEnable(
   number mInst,
   number axis,
   number enabel)
```

Description: Set the master soft limit enable flag for the specified axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axis	The axis ID.
enable	1 for enable

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Soft limits can be enabled on a per axis basis. This is the master soft limit enable for the axis. It overrides the soft limit enable set by mcSoftLimitSetState().

Usage:

```
// Diable softlimits on axis 0
MINSTANCE mInst = 0;
mcAxisSetSoftlimitEnable(mInst, 0,0);
```

mcAxisSetSoftlimitMax

C/C++ Syntax:

```
int mcAxisSetSoftlimitMax(
  MINSTANCE mInst,
  int axisId,
  double max);
```

LUA Syntax:

```
rc = mc.mcAxisSetSoftlimitMax(
  number mInst,
  number axisId,
  number max);
```

Description: Set the softlimit maximum for the axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
max	The maximum poisition of the axis in machine coordinates.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Set the softlimit maximum for the axis by drilling down to each motor and setting it max softlimit based on the counts per unit and max sent (max * countsperunit)

Usage:

```
int mInst=0;
```

```
int axis = Y_AXIS;
double max = 20;
mcAxisGetSoftlimitMax( mInst, axis, max);// Set the max distance of the softlimi
```

mcAxisSetSoftlimitMin

C/C++ Syntax:

```
int mcAxisSetSoftlimitMin(
  MINSTANCE mInst,
  int axisId,
  double min);
```

LUA Syntax:

```
rc = mc.mcAxisSetSoftlimitMin(
  number mInst,
  number axisId,
  number min);
```

Description: Set the softlimit minimum for the axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
min	The minimum position of the axis in machine coordinates.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Set the softlimit minimum for the axis by drilling down to each motor and setting it main softlimit based on the counts per unit and min sent (min * countsperunit)

Usage:

```
int mInst=0;
```

```
int axis = Y_AXIS;
double min = 20;
mcAxisGetSoftlimitMin( mInst, axis, min);// Set the min distance of the softlimit
```

mcAxisSetSpindle

C/C++ Syntax:

```
int mcAxisSetSpindle(
  MINSTANCE mInst,
  int axisId,
  BOOL spindle);
```

LUA Syntax:

```
rc = mc.mcAxisSetSpindle(
  number mInst,
  number axisId,
  number spindle)
```

Description: Flag or unflag and axis as controlling a spindle.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis ID.
spindle	The spindle flag.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:

Setting the current axis as an axis controlling a spindle has the effect of negating this flag for all other axes.
Only out of band axes can be falgged as controlling a spindle.

Usage:

```
// Set axis 6 as a spindle axis.
MINSTANCE mInst = 0;
mcAxisSetSpindle(mInst, 6, true);
```

mcAxisSetVel

C/C++ Syntax:

```
int mcAxisSetVel(
 MINSTANCE mInst,
 int axis,
 double velocity);
```

Description: Not used at this time.

mcAxisUnmapMotor

C/C++ Syntax:

```
int mcAxisUnmapMotor(
  MINSTANCE mInst,
  int axisId,
  int motor);
```

LUA Syntax:

```
rc = mc.mcAxisUnmapMotor(
  number mInst,
  number axisId,
  number motor)
```

Description: Unmap the motor from the axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The ID identifying the axis from which to unmap the motor.
motor	The ID identifying the motor to be unmapped from the aixs.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motor** was not found.

Notes:
Used to remove a motor from an axis based on it's motor id.

Usage:

```
int mInst = 0;
// Remove motor6 from the Y axis.
mcAxisUnmapMotor(mInst, Y_AXIS, MOTOR6);
```

mcAxisUnmapMotors

C/C++ Syntax:

```
int mcAxisUnmapMotors(
  MINSTANCE mInst,
  int axisId);
```

LUA Syntax:

```
rc = mc.mcAxisUnmapMotors(
  number mInst,
  number axisId)
```

Description: Unmap all motors from the axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	Axis number to unmap motor from.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Used to remove all motors from an axis. This is used when an axis needs to have all it's child motors removed and setup with all new motors.

Usage:

```
int mInst = 0;
// Remove all motors from the Y axis.
mcAxisUnmapMotors(mInst, Y_AXIS);
```

mcAxisUnregister

C/C++ Syntax:

```
int mcAxisUnregister(
  MINSTANCE mInst,
  int axisId);
```

LUA Syntax:

```
rc = mc.mcAxisUnregister(
  number mInst,
  number axisId)
```

Description: Unregister an axis from the system.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis id.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:

Usage:

```
int mInst=0;
// Remove AXIS7 from frome the system.
mcAxisUnregister(mInst, AXIS7);
```

mcCntlCheckLicenseFeature

C/C++ Syntax:

```
int mcCntlCheckLicenseFeature(
  MINSTANCE  mInst,
  const char *licFile,
  const char *requirement,
  const char *feature);
```

LUA Syntax:

```
rc = mcCntlCheckLicenseFeature(
  number  mInst,
  string licFile,
  string requirement,
  string feature);
```

Description: Check a license file for a requirement and feature.

Parameters:

Parameter	Description
mInst	The controller instance.
licFile	The license file name.
requirement	A string defining the license requirement.
feature	A string defining the feature to check.

Returns: (

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	The **feature** parameter cannot be NULL.
MERROR_NODATA	The license service count not be initialized.
MERROR_LIC_BAD_ID	The license file PCID does not match the current PCID.

MERROR_LIC_REQUIREMENT_NOT_FOUND	The specified license requirement does not exist in the license file.
MERROR_LIC_FEATURE_NOT_FOUND	The specified feature does not exist in the license file.
MERROR_LIC_EXPIRED	The time limited license has expired.
MERROR_LIC_BAD_KEY	The license encryption key is invalid.

Notes:

If no explicit path is given for the **licFile** parameter (just a file name), the core looks for the license in the Licenses directory of the installation path. Otherwise, the fully qualified path is used.

Usage:

```
// Check the DarwinLic.dat file for feature M4_DARWIN
MINSTANCE mInst = 0;
int rc = mcCntlCheckLicenseFeature(mInst, "DarwinLic.dat", "NF_DARWIN", "M4_DARWI
if (rc != MERROR_NOERROR) {
 // Feature found!!!
 return;
}
```

mcCntlCleanup

C/C++ Syntax:

```
int mcCntlCleanup(MINSTANCE mInst);
```

LUA Syntax:

```
N/A
```

Description: Cleanup the core instance before shutdown.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
None.
// Cleanup core instance 0
MINSTANCE mInst = 0;
int rc = mcCntlCleanup(mInst);
```

mcCntlCloseGCodeFile

C/C++ Syntax:

```
int mcCntlCloseGCodeFile(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcCntlCloseGCodeFile(
  number mInst)
```

Description: Close Gocde file.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Used to close a Gcode file if the Gcode File needs to be edited

Usage:

```
int mInst=0;
mcCntlCloseGCodeFile(mInst); // Close the Gcode file in controller 0.
```

mcCntlCompileScripts

C/C++ Syntax:

```
int mcCntlCompileScripts(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcCntlCompileScripts(
  number mInst);
```

Description: Force scripts to compile (if the script plugin allows).

Parameters:

Parameter	Description
mInst	The controller instance.

Returns: _RETUNR(, the scripts could not be compiled.

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_NOW	The operation could not be completed at this time.

Notes:

Usage:

```
// Compaile all scripts
MINSTANCE mInst = 0;
int rc = mcCntlCompileScripts(mInst);
```

mcCntlConfigStart

C/C++ Syntax:

```
int mcCntlConfigStart(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcCntlConfigStart(
  number mInst);
```

Description: Put the control into the configure state.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_STATE	The control is in a state from which it cannot transition to the configure state.

Notes:
Upon entering the configure state, MSG_CONFIG_START is sent to the GUI and plugins.

Usage:

```
// Enter the configure state
MINSTANCE mInst = 0;
if (mcCntlConfigStart(mInst) == MERRROR_NOERROR) {
 // Successfully entered the configure state!
}
```

mcCntlConfigStop

C/C++ Syntax:

```
int mcCntlConfigStop(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcCntlConfigStop(
  number mInst);
```

Description: Leave the configure state.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_STATE	The control was not in the configure state.

Notes:
Upon leaving the congifure state, MSG_CONFIG_END is sent to the GUI and plugins.

Usage:

```
// Leave the configure state.
MINSTANCE mInst = 0;
int rc = mcCntlConfigStop(mInst);
if (rc == MERROR_NOERROR) {
 // Successfully exited the configure state.
}
```

mcCntlCycleStart

C/C++ Syntax:

```
int mcCntlCycleStart(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcCntlCycleStart(
  number mInst)
```

Description: Start the Gcode file running.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_NOW	The operation could not be completed at this time.

Notes:
Is also used to restart after a feedhold.

Usage:

```
int mInst=0;
// Start controller 0's Gcode file.
mcCntlCycleStart(mInst);
```

mcCntlCycleStop

C/C++ Syntax:

```
int mcCntlCycleStop(
 MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcCntlCycleStop(
  number mInst)
```

Description: Stop a Gcode file that is running.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_NOW	The operation could not be completed at this time.

Notes:
Feedhold needs to be used to pause the file. CMD_STOP_EXECUTION is sent to all plugins to tell the the Stop event has been requested.

Usage:

```
int mInst=0;
// Stop controller 0's Gcode file.
mcCntlCycleStop(mInst);
```

mcCntlDryRunToLine

C/C++ Syntax:

```
int mcCntlDryRunToLine(
  MINSTANCE mInst,
  int line);
```

LUA Syntax:

```
rc = mc.mcCntlDryRunToLine(
  number mInst,
  number line)
```

Description: Process G code forward to a given line without motion.

Parameters:

Parameter	Description
mInst	The controller instance.
line	An integer specifying the G code line number.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_ENABLED	The control is not enabled.
MERROR_NOT_NOW	The operation could not be completed at this time.
It was not possible to start the dry run at this time.	

Notes:
Once the dry run is started, the function returns immediately. To wait on the dry run, you must check the state of the control.

Usage:

```
// Dry run to line 38
MINSTANCE mInst = 0;
```

```
int rc = mcCntlDryRunToLine(mInst, 38);
```

mcCntlEStop

C/C++ Syntax:

```
int mcCntlEStop(
 MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcCntlEStop(
 number mInst)
```

Description: Disables the control and optionally dereferences all axes.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
This function is not a subsitute for a hard wired e-stop circuit! This is a convenience function that is designed to put the contorol into a default state to be used after the machine has already been stopped by the real e-stop control circuit.

Usage:

```
// Put the control in the default state when e-stop is asserted.
MINSTANCE mInst = 0;
int rc = mcCntlEStop(mInst);
```

mcCntlEnable

C/C++ Syntax:

```
int mcCntlEnable(
 MINSTANCE mInst,
 BOOL state);
```

LUA Syntax:

```
rc = mc.mcCntlEnable(
 number mInst,
 number state)
```

Description: Enable or disable the control.

Parameters:

Parameter	Description
mInst	The controller instance.
state	A BOOL specifying the desired control state. TRUE for enabled

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_NOW	The operation could not be completed at this time.
The ISIG_EMERGENCY signal is asserted.	

Notes:
If ISIG_EMERGENCY is asserted, the e-stop condition must be cleared.

Usage:

```
// Enable the control.
MINSTANCE mInst = 0;
int rc = mcCntlEnable(mInst, TRUE);
```

mcCntlFeedHold

C/C++ Syntax:

```
int mcCntlFeedHold(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcCntlFeedHold(
  number mInst)
```

Description: Pause the motion of a Gcode block.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_NOW	The operation could not be completed at this time.

Notes:
Will not effect out of band axis or jogging.

Usage:

```
int mInst = 0;
int rc;
rc = mcCntlFeedHold(mInst); // Pause the motion of controller 0.
if (rc == MERROR_NOERROR) {
 // Feed hold was successful.
}
```

mcCntlFeedHoldState

C/C++ Syntax:

```
int mcCntlFeedHoldState(
  MINSTANCE mInst,
  BOOL *InFeedHold);
```

LUA Syntax:

```
InFeedHold, rc = mc.mcCntlFeedHoldState(
    number mInst)
```

Description: Get the state of Feedhold.

Parameters:

Parameter	Description
mInst	The controller instance.
InFeedHold	The address of a BOOL to receive the status of Feedhold (TRUE or FALSE).

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Not used at this time.

Usage:

```
int mInst = 0;
int rc;
BOOL fhState;
rc = mcCntlFeedHoldState(mInst, &fhState;); // Pause the motion of controller 0.
if (rc == MERROR_NOERROR && fhState == TRUE) {
 // The control is in Feedhold.
}
```

mcCntlGcodeExecute

C/C++ Syntax:

```
int mcCntlGcodeExecute(
  MINSTANCE mInst,
  const char *commands);
```

LUA Syntax:

```
rc = mc.mcCntlGcodeExecute(
  number mInst,
  string commands)
```

Description: Execute G code as a unit of work.

Parameters:

Parameter	Description
mInst	The controller instance.
commands	A string buffer containing G code.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_NOW	The operation could not be completed at this time.
MERROR_NOT_COMPILED	The macro scripts could not be compiled.

Notes:

If the control is in the idle state, this function will execute the G code in MDI mode.

It is important to note that this function does not wait for the G code to complete. It is an error to execute G code with this function one line after another. Instead, the lines should be combined into one string separeated with newline characters (\n).

Usage:

```
// Execute spindle stop and rapid to 0, 0.
MINSTANCE mInst = 0;
int rc;
rc = mcCntlGcodeExecute(mInst, "M05\nG00 X0 Y0");
if (rc == MERROR_NOERROR) {
 // The G code was submitted for processing.
// However, the function return immediately and does not wait on the G code to f
}
```

mcCntlGcodeExecuteWait

C/C++ Syntax:

```
int mcCntlGcodeExecuteWait(
  MINSTANCE mInst,
  const char *commands);
```

LUA Syntax:

```
rc = mc.mcCntlGcodeExecute(
  number mInst,
  string commands)
```

Description: Execute G code as a unit of work and wait until it is complete.

Parameters:

Parameter	Description
mInst	The controller instance.
commands	A string buffer containing G code.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_NOW	The operation could not be completed at this time.
MERROR_NOT_COMPILED	The macro scripts could not be compiled.

Notes:

If the control is in the idle state, this function will execute the G code in MDI mode.

Usage:

```
// Execute spindle stop and rapid to 0, 0.
MINSTANCE mInst = 0;
int rc;
rc = mcCntlGcodeExecute(mInst, "M05\nG00 X0 Y0");
if (rc == MERROR_NOERROR) {
```

```
 // The G code was submitted for processing and has completed.
}
```

mcCntlGcodeInterpGetData

C/C++ Syntax:

```
int mcCntlGcodeInterpGetData(
  MINSTANCE mInst,
  interperter_t *data);
```

LUA Syntax:

```
N/A
```

Description: Retrieves the current G code interpreter data.

Parameters:

Parameter	Description
mInst	The controller instance.
data	The address of a Interpreter_info struct.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

```
struct Interperter_info {
 double ModalGroups[MC_MAX_GROUPS];
 double FeedRate;
 double SpindleSpeed;
 int SpindleDirection;
 BOOL Mist;
 BOOL Flood;
 int ToolNumber;
 int HeightRegister;
 int DiaRegister;
};
typedef struct Interperter_info interperter_t;
```

Usage:

```
// Get the interpreter information.
MINSTANCE mInst = 0;
```

```
interperter_t data;
int rc;
rc = mcCntlGcodeInterpGetData(mInst, &data;);
if (rc == MERROR_NOERROR) {
 // Success!
}
```

mcCntlGcodeInterpGetPos

C/C++ Syntax:

```
int mcCntlGcodeInterpGetPos(
  MINSTANCE mInst,
  int axisId,
  double *pos);
```

LUA Syntax:

```
pos, rc = mc.mcCntlGcodeInterpGetPos(
  number mInst,
  number axisId)
```

Description: Retrieves the current G code interpreter position for the given axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	An integer specifying the axis.
pos	An address of a double to receive the axis' position.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
It is important to not that the look ahead buffer will affect what value is returned and it may not be the current position.

Usage:

```
// Get the current buffered X axis position.
MINSTANCE mInst = 0;
double pos;
int rc;
rc = int mcCntlGcodeInterpGetPos(mInst, X_AXIS, &pos;);
if (rc == MERROR_NOERROR) {
 // Success!
}
```

mcCntlGetBlockDelete

C/C++ Syntax:

```
int mcCntlGetBlockDelete(
  MINSTANCE mInst,
  int deleteId,
  BOOL *val);
```

LUA Syntax:

```
val, rc = mc.mcCntlGetBlockDelete(
  number mInst,
  number deleteId)
```

Description:

Parameters:

Parameter	Description
mInst	The controller instance.
deleteId	An integer specifying the block delete index.
val	The address of a BOOL to revceive the block delete state. (TRUE is block delete on

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

Up to 10 block deletes can be used.

Usage:

```
// Is block delete 0 on?
MINSTANCE mInst = 0;
BOOL val;
int rc = mcCntlGetBlockDelete(mInst, 0, &val;);
if (rc == MERROR_NOERROR) {
 if (val == TRUE) {
  // Is block delete 0 is on!
 } else {
```

```
  // Is block delete 0 is off!
 }
}
```

mcCntlGetBuild

C/C++ Syntax:

```
int mcCntlGetBuild(
  MINSTANCE mInst,
  char *buf,
  size_t bufsize);
```

LUA Syntax:

```
build, rc = mc.mcCntlGetBuild(
  number mInst)
```

Description: Retrieves the core build number.

Parameters:

Parameter	Description
mInst	The controller instance.
buf	A string buffer that receives the build number.
bufsize	The length of the supplied string buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

Usage:

```
// Get the core's build number
MINSTANCE mInst = 0;
char buildBuf[80];
int rc = mcCntlGetBuild(mInst, buildBuf, sizeof(buildBuf));
if (rc == MERROR_NOERROR) {
 printf("The current build is %s.n", buildBuf);
}
```

mcCntlGetComputerID

C/C++ Syntax:

```
int mcCntlGetComputerID(
  MINSTANCE mInst,
  char *buf,
  size_t bufSize);
```

LUA Syntax:

```
buf, rc = mc.mcCntlGetComputerID(
  number mInst)
```

Description: Retrieve the computer ID(s) for the current host PC.

Parameters:

Parameter	Description
mInst	The controller instance.
buf	A string buffer to receive the ID(s).
bufSize	The size of the string buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Depending on the number of Ethernet cards in the PC, this functions may return more than one ID. If more than one ID is returned, they will be delimited by tab characters.

Usage:

```
// Get the ID(s) for this host
MINSTANCE mInst = 0;
int rc = mcCntlGetComputerID(mInst, buf, sizeof(buf));
if (rc == MERROR_NOERROR) {
 // Parse buf to get the ID(s)
}
```

mcCntlGetCoolantDelay

C/C++ Syntax:

```
int mcCntlGetCoolantDelay(
  MINSTANCE mInst,
  double *secs);
```

LUA Syntax:

```
sec, rc = mc.mcCntlGetCoolantDelay(
  number mInst)
```

Description: Get the coolant delay time in seconds.

Parameters:

Parameter	Description
mInst	The controller instance.
secs	an address of a double to receive the coolant delay.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Get the coolant delay.
MINSTANCE mInst = 0;
double secs;
int rc = mcCntlGetCoolantDelay(mInst, &secs;);
if (rc == MERROR_NOERROR) {
 // Success!
}
```

mcCntlGetCwd

C/C++ Syntax:

```
int mcCntlGetCwd(
  MINSTANCE mInst,
  char *buf,
  size_t bufSize);
```

LUA Syntax:

```
buf, rc = mc.mcCntlGetCwd(
  number mInst)
```

Description: Retrieves the core's current working directory.

Parameters:

Parameter	Description
mInst	The controller instance.
buf	A string buffer to receive the ID(s).
bufSize	The size of the string buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

Usage:

```
// Get current working directory.
MINSTANCE mInst = 0;
char curDir[255];
int rc = mcCntlGetCwd(mInst, curDir, sizeof(curDir));
if (rc == MERROR_NOERROR) {
 // Success!
}
```

mcCntlGetDiaMode

C/C++ Syntax:

```
int mcCntlGetDiaMode(
  MINSTANCE mInst,
  BOOL *dia);
```

LUA Syntax:

```
dia, rc = mc.mcCntlGetDiaMode(
  number mInst)
```

Description: Retrieve the lathe diameter mode.

Parameters:

Parameter	Description
mInst	The controller instance.
dia	The address of a BOOL that receives the diameter mode. (TRUE is diameter mode

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Only valid when the control is in lathe (turn) mode.

Usage:

```
// Get the lathe diameter mode.
MINSTANCE mInst = 0;
BOOL Dia;
int rc = mcCntlGetDiaMode(mInst, &Dia;);
if (rc == MERROR_NOERROR) {
 // Success!
}
```

mcCntlGetDistToGo

C/C++ Syntax:

```
int mcCntlGetDistToGo(
 MINSTANCE mInst,
 int axisId,
 double *togo);
```

LUA Syntax:

```
togo, rc = int mcCntlGetDistToGo(
 number mInst,
 number axisId)
```

Description: Returns the distance from the end point of a move in current block.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis from which to to get the distance.
togo	A pointer to a double to receive the distance left in current move.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.
MERROR_NOT_NOW	The operation could not be completed at this time.

Notes:
Only used when running G code.

Usage:

```
int mInst=0;
int axis = Y_AXIS;
```

```
double togo=0;
mcCntlGetDistToGo(mInst, axis, &togo;);
```

mcCntlGetFRO

C/C++ Syntax:

```
int mcCntlGetFRO(
  MINSTANCE mInst,
  double *percent);
```

LUA Syntax:

```
percent, rc = mc.mcCntlGetFRO(
  number mInst)
```

Description: Get the current Feed Rate Override percentage.

Parameters:

Parameter	Description
mInst	The controller instance.
percent	The Address of a double to receive the feed rate override percentage.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Get the current FRO
MINSTANCE mInst = 0;
double fro;
int rc = mcCntlGetFRO(mInst, &fro;);
if (rc == MERROR_NOERROR) {
 // Success!
}
```

mcCntlGetGcodeFileName

C/C++ Syntax:

```
int mcCntlGetGcodeFileName(
  MINSTANCE mInst,
  char *buf,
  size_t bufSize);
```

LUA Syntax:

```
buf, rc = mc.mcCntlGetGcodeFileName(
  number mInst)
```

Description: Retrieve he current G code file name.

Parameters:

Parameter	Description
mInst	The controller instance.
buf	A string buffere to receive the G code file name.
bufSize	The length of the string buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
An empty string is returned if there is no G code file loaded.

Usage:

```
// Get the currently loaded G code file name.
MINSTANCE mInst = 0;
char gCodeFileName[255];
int rc = mcCntlGetGcodeFileName(mInst, gCodeFileName, sizeof(gCodeFileName));
if (rc == MERROR_NOERROR) {
 // Success.
}
```

mcCntlGetGcodeLine

C/C++ Syntax:

```
int mcCntlGetGcodeLine(
  MINSTANCE mInst,
  int LineNumber,
  char *buf,
  long bufSize);
```

LUA Syntax:

```
buff, rc = mc.mcCntlGetGcodeLine(
  number mInst,
  number LineNumber)
```

Description: Get the line of Gcode from the loaded gcode file.

Parameters:

Parameter	Description
mInst	The controller instance.
LineNumber	The desired line number from the Gcode file.
buf	The address of a character buffer that recieves the Gcode line.
bufSize	The size of the receiving character buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**LineNumber** is out of range

Notes:
Used to get the gocde file lines to display

Usage:

```
int mInst=0;
int LineNumber = 5;
char gline[128];
```

```
gline[0] = '0';
mcCntlGetGcodeLine(mInst, LineNumber, gline , 128); // Get line number 5 from the
```

mcCntlGetGcodeLineCount

C/C++ Syntax:

```
int mcCntlGetGcodeLineCount(
  MINSTANCE mInst,
  double *count);
```

LUA Syntax:

```
count, rc = mc.mcCntlGetGcodeLineCount(
  number mInst)
```

Description: Get the number of lines in the Gcode file.

Parameters:

Parameter	Description
mInst	The controller instance.
count	The address of a double to receive the number of lines in the G code file.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
The number of lines in the file may not be the number of blocks or moves to be done because of sub routines and canned cycles.

Usage:

```
int mInst=0;
double count=0;
mcCntlGetGcodeLineCount(mInst, &count;);
```

mcCntlGetGcodeLineNbr

C/C++ Syntax:

```
int mcCntlGetGcodeLineNbr(
  MINSTANCE mInst,
  double *val);
```

LUA Syntax:

```
val, rc = mcCntlGetGcodeLineNbr(
  number mInst)
```

Description: Returns the line number of the current move being made.

Parameters:

Parameter	Description
mInst	The controller instance.
val	The address of a double to receive the G code line number.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
This function will return the next line that will be run if G code is not running.

Usage:

```
int mInst=0;
double dline=0;
mcCntlGetGcodeLineNbr(mInst, &dline;); //Get the current running line number
```

mcCntlGetLastError

C/C++ Syntax:

```
int mcCntlGetLastError(
  MINSTANCE mInst,
  char *buf,
  size_t bufSize);
```

LUA Syntax:

```
buf, rc = mc.mcCntlGetLastError(
  MINSTANCE mInst)
```

Description: Retrieve the last error/status message from the core.

Parameters:

Parameter	Description
mInst	The controller instance.
buf	A string buffer to receive the next error message on the error message stack.
bufSize	The length of the string buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**buf** was NULL or **bufSize** was 0.
MERROR_NODATA	No message was available.

Notes:
Calling this function removes the returned message from the stack.

Usage:

```
int mInst=0;
char error[128];
mcCntlGetLastError(mInst, error, 128); // Receives the last error messge for cont
```

mcCntlGetLastLogMsg

C/C++ Syntax:

```
int mcCntlGetLastLogMsg(
  MINSTANCE mInst,
  char *buf,
  size_t bufSize);
```

LUA Syntax:

```
buf, rc = mc.mcCntlGetLastLogMsg(
  number mInst)
```

Description: Retrieve the last log message from the core.

Parameters:

Parameter	Description
mInst	The controller instance.
buf	A string buffer to receive the next log message on the log message stack.
bufSize	The length of the string buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**buf** was NULL or **bufSize** was 0.
MERROR_NODATA	No log message was available.

Notes:
Calling this function removes the returned message from the log stack.

Usage:

```
int mInst=0;
char msg[128];
mcCntlGetLastLogMsg(mInst, msg, 128); // Receives the last log messge for contro
```

mcCntlGetLicenseData

C/C++ Syntax:

```
int mcCntlGetLicenseData(
  MINSTANCE mInst,
  int index,
  char *buf,
  long bufSize);
```

LUA Syntax:

```
buf, rc = mc.mcCntlGetLicenseData(
  number mInst,
  number index)
```

Description: Retrieves the license data for the specified index.

Parameters:

Parameter	Description
mInst	The controller instance.
index	The index of the license data to retrieve.
buf	The address of a character buffer that recieves the Gcode line.
bufSize	The size of the receiving character buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**index** is out of range
MERROR_NODATA	There is no license data for the specidied index.

Notes:

Usage:

```
// Get the license data at index 1.
MINSTANCE mInst = 0;
```

```
long len;
char *data;
long dataLen = 0;
int index = 1;
int rc = mcCntlGetLicenseDataLen(mInst, index, &dataLen;);
if (rc == MERROR_NOERROR && dataLen > 0) {
 data = (char *)malloc(dataLen + 1);
 memset(data, 0, dataLen + 1);
 rc = mcCntlGetLicenseData(mInst, index, data, dataLen);
 if (rc == MERROR_NOERROR) {
  // Success!
 }
}
```

mcCntlGetLicenseDataLen

C/C++ Syntax:

```
int mcCntlGetLicenseDataLen(
  MINSTANCE mInst,
  int index,
  long *bufSize);
```

LUA Syntax:

```
bufSize, rc = mc.mcCntlGetLicenseData(
  number mInst,
  number index)
```

Description: Retrieves the license data length for the specified index.

Parameters:

Parameter	Description
mInst	The controller instance.
index	The index of the license data to retrieve.
bufSize	The address of a long to receive the data length..

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**index** is out of range
MERROR_NODATA	There is no license data for the specidied index.

Notes:

Usage:

```
// Get the license data at index 1.
MINSTANCE mInst = 0;
long len;
char *data;
long dataLen = 0;
```

```
int index = 1;
int rc = mcCntlGetLicenseDataLen(mInst, index, &dataLen;);
if (rc == MERROR_NOERROR && dataLen > 0) {
 data = (char *)malloc(dataLen + 1);
 memset(data, 0, dataLen + 1);
 rc = mcCntlGetLicenseData(mInst, index, data, dataLen);
 if (rc == MERROR_NOERROR) {
  // Success!
 }
}
```

mcCntlGetLicenseModules

C/C++ Syntax:

```
int mcCntlGetLicenseModules(
  MINSTANCE mInst,
  unsigned long long *modules)
```

LUA Syntax:

```
N/A
```

Description: Retrieve the 64 bit number comprising the licensed modules.

Parameters:

Parameter	Description
mInst	The controller instance.
modules	The address of a unsigned lon long to receive the licensed modules..

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Retrieve the licensed modules.
MINSTANCE mInst = 0;
unsigned long long modules = 0;
int rc = mcCntlGetLicenseModules(mInst, &modules;);
```

mcCntlGetLocalVar

int mcCntlGetLocalVar(MINSTANCE mInst, HMCVARS hVars, int varNumber, doubl *retval);

LUA Syntax:

```
retval, rc = mc.mcCntlGetLocalVar(
  number mInst,
  number hVars,
  number varNumber)
```

Description:

Retrieve the specified local variable.

Parameters:

Parameter	Description
mInst	The controller instance.
hVars	The handle to the loacl variable stack.
varNumber	The index of the local variable flage to retrieve.
retval	An address of a double integer to receive the value of the specified local variable.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**varNumber** is out of range or **retval** is NULL.

Notes:

This is primarily a script function.

Usage:

```
-- Get local variables.
function m700(hVars)
 local inst = mc.mcGetInstance() -- Get the current instance
 local nilPoundVar = mc.mcCntlGetPoundVar(inst,0)
 local message = ""
 if hVars ~= nil then
  local flag, retval, rc
```

```
  flag, rc = mc.mcCntlGetLocalVarFlag(inst, hVars, mc.SV_A)
  if (flag == 1) then
   retval, rc = mc.mcCntlGetLocalVar(inst, hVars, mc.SV_A)
   if rc == mc.MERROR_NOERROR then
    message = message .. "A" .. ":" .. tostring(retval) .. ", "
   end
  end
  flag, rc = mc.mcCntlGetLocalVarFlag(inst, hVars, mc.SV_A)
  if (flag == 1) then
   retval, rc = mc.mcCntlGetLocalVar(inst, hVars, mc.SV_B)
   if rc == mc.MERROR_NOERROR then
    message = message .. "B" .. ":" .. tostring(retval)
   end
  end
  mc.mcCntlSetLastError(inst, message)
 end
end

if (mc.mcInEditor() == 1) then
    m700(nil) -- We can't test this in the editor!
end
```

mcCntlGetLocalVarFlag

C/C++ Syntax:

```
int mcCntlGetLocalVarFlag(
  MINSTANCE mInst,
  HMCVARS hVars,
  int varNumber,
  int *retval);
```

LUA Syntax:

```
retval, rc = mc.mcCntlGetLocalVarFlag(
  number mInst,
  number hVars,
  number varNumber)
```

Description: Retrieve the specified local variable.

Parameters:

Parameter	Description
mInst	The controller instance.
hVars	The handle to the loacl variable stack.
varNumber	The index of the local variable flage to retrieve.
retval	An address of an integer to receive the value of the specified local variable flag.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**varNumber** is out of range or **retval** is NULL.

Notes:
This is primarily a script function.

Usage:

```
-- Get local variables.
```

```
function m700(hVars)
 local inst = mc.mcGetInstance() -- Get the current instance
 local nilPoundVar = mc.mcCntlGetPoundVar(inst,0)
 local message = ""
 if hVars ~= nil then
  local flag, retval, rc
  flag, rc = mc.mcCntlGetLocalVarFlag(inst, hVars, mc.SV_A)
  if (flag == 1) then
   retval, rc = mc.mcCntlGetLocalVar(inst, hVars, mc.SV_A)
   if rc == mc.MERROR_NOERROR then
    message = message .. "A" .. ":" .. tostring(retval) .. ", "
   end
  end
  flag, rc = mc.mcCntlGetLocalVarFlag(inst, hVars, mc.SV_A)
  if (flag == 1) then
   retval, rc = mc.mcCntlGetLocalVar(inst, hVars, mc.SV_B)
   if rc == mc.MERROR_NOERROR then
    message = message .. "B" .. ":" .. tostring(retval)
   end
  end
  mc.mcCntlSetLastError(inst, message)
 end
end

if (mc.mcInEditor() == 1) then
    m700(nil) -- We can't test this in the editor!
end
```

mcCntlGetLogging

C/C++ Syntax:

```
int mcCntlGetLogging(
  MINSTANCE mInst,
  BOOL *enabled);
```

LUA Syntax:

```
enabled, rc = mc.mcCntlGetLogging(
  number mInst)
```

Description: Determine if logging is enabled.

Parameters: (enabled, The address of a BOOL to receive the logging state.

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**enabled** is NULL.

Notes:
None.

Usage:

```
// Check is logging is enabled.
MINSTANCE mInst = 0;
BOOL enabled = FALSE;
int rc = mcCntlGetLogging(mInst, &enabled;);
if (rc == MERROR_NOERROR && enabled == TRUE) {
 // Logging is enabled!
}
```

mcCntlGetMachDir

C/C++ Syntax:

```
int mcCntlGetMachDir(
  MINSTANCE mInst,
  char *buf,
  size_t bufSize);
```

LUA Syntax:

```
buf, rc = int mcCntlGetMachDir(
  number mInst)
```

Description: Retrieves the core installation directory.

Parameters:

Parameter	Description
mInst	The controller instance.
buf	The address of a character buffer that recieves the core installation directory.
bufSize	The size of the receiving character buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**buf** is NULL

Notes:
None.

Usage:

```
// Get the Mach core installation directory.
MINSTANCE mInst = 0;
char instDir[255];
memset(instDir, 0, sizeof(instDir);
int rc = mcCntlGetMachDir(mInst, instDir, sizeof(instDir));
```

mcCntlGetMistDelay

C/C++ Syntax:

```
int mcCntlGetMistDelay(
  MINSTANCE mInst,
  double *secs);
```

LUA Syntax:

```
secs, rc = mcCntlGetMistDelay(
  number mInst)
```

Description: Get the mist delay time in seconds.

Parameters:

Parameter	Description
mInst	The controller instance.
secs	an address of a double to receive the mist delay.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Get the mist delay.
MINSTANCE mInst = 0;
double secs;
int rc = mcCntlGetMistDelay(mInst, &secs;);
if (rc == MERROR_NOERROR) {
 // Success!
}
```

mcCntlGetModalGroup

C/C++ Syntax:

```
int mcCntlGetModalGroup(
  MINSTANCE mInst,
  int group,
  double *val);
```

LUA Syntax:

```
val, rc = mc.mcCntlGetModalGroup(
  number mInst,
  number group)
```

Description: Get the modal group code for the specified group.

Parameters:

Parameter	Description
mInst	The controller instance.
group	A integer specifying the modal group.
val	The address of a double to receive the modal group value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**val** is NULL or **group** is out of range.

Notes:
If the modal group is not active, -1 is returned.

Usage:

```
// Get the modal code for modal group 1.
MINSTANCE mInst = 0;
int rc = mcCntlGetModalGroup(mInst, 1);
if (rc == MERROR_NOERROR) {
 // Success!
}
```

mcCntlGetMode

C/C++ Syntax:

```
int mcCntlGetMode(
  MINSTANCE mInst,
  int *mode);
```

LUA Syntax:

```
mode, rc = mcCntlGetMode(
  number mInst)
```

Description: Get the current controller mode.

Parameters:

Parameter	Description
mInst	The controller instance.
mode	The address of an integer to receive the control mode. MC_MODE_MILL, MC_MODE_LATHE_DIA, MC_MODE_LATHE_RAD, MC_MODE_TANGENTIAL

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Get the control mode.
MINSTANCE mInst = 0;
int mode;
int rc = mcCntlGetMode(mInst, &mode;);
if (rc == MERROR_NOERROR) {
 // Success!
}
```

mcCntlGetOffset

C/C++ Syntax:

```
int mcCntlGetOffset(
  MINSTANCE mInst,
  int axisId,
  int type,
  double *offset);
```

LUA Syntax:

```
offset, rc = mc.mcCntlGetOffset(
  number mInst,
  number axisId,
  number type)
```

Description: Retrieve the offset specified by type for the axis specified by axisId

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	An Interger specifying the axis ID. (X_AXIS, Y_AXIS, Z_AXIS)
type	An Interger specifying the offset type. (MC_OFFSET_FIXTURE, MC_OFFSET_AXIS

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

Usage:

```
//
MINSTANCE mInst = 0;
```

mcCntlGetOptionalStop

C/C++ Syntax:

```
int mcCntlGetOptionalStop(
  MINSTANCE mInst,
  BOOL *stop);
```

LUA Syntax:

```
stop, rc = mc.mcCntlGetOptionalStop(
  number mInst)
```

Description: Retrieve the state of optional stop.

Parameters:

Parameter	Description
mInst	The controller instance.
stop	The address of a BOOL to contain the optional stop state.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**stop** is NULL.

Notes:
None.

Usage:

```
// See if optional stop is in effect.
MINSTANCE mInst = 0;
BOOL opStop = FALSE;
int rc = mcCntlGetOptionalStop(mInst, &opStop;);
if (rc == MERROR_NOERROR) {
 if (opStop == TRUE) {
  // Optional Stop is in effect!
 } else {
  // Optional Stop is not in effect!
 }
```

}

mcCntlGetPoundVar

C/C++ Syntax:

```
int mcCntlGetPoundVar(
  MINSTANCE mInst,
  int param,
  double *value);
```

LUA Syntax:

```
value, rc = mc.mcCntlGetPoundVar(
  number mInst,
  number param)
```

Description: Get the value of a system variable (pound variable).

Parameters:

Parameter	Description
mInst	The controller instance.
param	An integer specifying a system variable.
*value	The address of a double to receive the value of the system variable.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**param** is out of range.

Notes:

This is a good tool to assist in debugging G code files with system variable (pound variable) programing.

Note that system variables can reflect values in the future because of the look ahead buffer.

Usage:

```
int mInst=0;
```

```
double PoundVar=50;
dobule Value=0;
;// Get the value of #50.
mcCntlGetPoundVar(mInst, PoundVar, &Value;);
```

mcCntlGetRRO

C/C++ Syntax:

```
int mcCntlGetRRO(
  MINSTANCE mInst,
  double *percent);
```

LUA Syntax:

```
percent, rc = mc.mcCntlGetRRO(
  number mInst)
```

Description: Get the current Feed Rate Override percentage.

Parameters:

Parameter	Description
mInst	The controller instance.
percent	The Address of a double to receive the rapid rate override percentage.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Get the current RRO
MINSTANCE mInst = 0;
double rro;
int rc = mcCntlGetRRO(mInst, &rro;);
if (rc == MERROR_NOERROR) {
 // Success!
}
```

mcCntlGetRunTime

C/C++ Syntax:

```
int mcCntlGetRunTime(
  MINSTANCE mInst,
  double *time);
```

LUA Syntax:

```
int mcCntlGetRunTime(MINSTANCE mInst, double *time)
```

Description: Get the control run time in seconds.

Parameters:

Parameter	Description
mInst	The controller instance.
time	The address of a double to receive the run time in seconds.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**time** is NULL.

Notes:
None.

Usage:

```
// Get the control run time in seconds.
MINSTANCE mInst = 0;
double time;
int rc = mcCntlGetRunTime(mInst, &time;);
if (rc == MERROR_NOERROR) {
 // Success!
}
```

mcCntlGetSingleBlock

C/C++ Syntax:

```
int mcCntlGetSingleBlock(
  MINSTANCE mInst,
  BOOL *sbState);
```

LUA Syntax:

```
sbState, rc = mc.mcCntlGetSingleBlock(
  number mInst)
```

Description: Get the state of Single block.

Parameters:

Parameter	Description
mInst	The controller instance.
sbState	The address of a BOOL to receive the state of single block. (MC_ON/TRUE, MC_OFF/FALSE)

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**sbState** is NULL.

Notes:
Used to display the state of single block to the user.

Usage:

```
// Get the state of single block.
int mInst=0;
BOOL IsOn = FASLE;
mcCntlGetSingleBlock(mInst, &IsOn;);
```

mcCntlGetState

C/C++ Syntax:

```
int mcCntlGetState(
  MINSTANCE mInst,
  mcState *state);
```

LUA Syntax:

```
mcState, rc = mc.mcCntlGetState(
  MINSTANCE mInst)
```

Description: Get the current controller state.

Parameters:

Parameter	Description
mInst	The controller instance.
state	The address of a mcState variable to receive the controller state.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**state** is NULL.

Notes:
None.

Usage:

```
// Get the state of controller instance 0.
MINSTANCE mInst = 0;
mcState state;
char stateName[80];
int rc = mcCntlGetState(mInst, &state;);
if (rc == MERROR_NOERROR) {
 // Success!
 rc = mcCntlGetStateName(mInst, state, stateName, sizeof(stateName));
}
```

mcCntlGetStateName

C/C++ Syntax:

```
int mcCntlGetStateName(
  MINSTANCE mInst,
  mcState state,
  char *buf,
  size_t bufSize);
```

LUA Syntax:

```
buf, rc = mc.mcCntlGetStateName(
  number mInst,
  number state)
```

Description: Retrieve the state name for the given mcState.

Parameters:

Parameter	Description
mInst	The controller instance.
state	A mcState variable specifying the state.
buf	A string buffer to receive the state name..
bufSize	The length of the string buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**buf** is NULL or **bufSize** is less than or equal to zero.
MERROR_INVALID_STATE	**state** is out of range.

Notes:
None.

Usage:

```
// Get the state name for controller instance 0.
```

```
MINSTANCE mInst = 0;
mcState state;
char stateName[80];
int rc = mcCntlGetState(mInst, &state;);
if (rc == MERROR_NOERROR) {
 // Success!
 rc = mcCntlGetStateName(mInst, state, stateName, sizeof(stateName));
}
```

mcCntlGetStats

C/C++ Syntax:

```
int mcCntlGetStats(
  MINSTANCE mInst,
  mstats_t *stats);
```

LUA Syntax:

```
N/A
```

Description: Retrieve the control statistics.

Parameters:

Parameter	Description
mInst	The controller instance.
stats	The address of a mstats_t variable to receive the statistics.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**stats** is NULL.

Notes:

```
struct mstats {
 int cannon_buf_depth;
 int la_cannon_buf_depth;
 double totalTime;
 double sessionTime;
 double spindleTime;
 long m3count;
 long m4count;
 long m6count;
 double xDistance;
 double yDistance;
 double zDistance;
 double aDistance;
 double bDistance;
```

```
 double cDistance;
};
typedef struct mstats mstats_t;
```

Usage:

```
// Get controller statistics.
MINSTANCE mInst = 0;
mstats_t stats;
int rc = mcCntlGetStats(mInst, &stats;);
if (rc == MERROR_NOERROR) {
 printf("M3 count = %dn", stats.m3Count);
}
```

mcCntlGetToolOffset

C/C++ Syntax:

```
int mcCntlGetToolOffset(
  MINSTANCE mInst,
  int axisId,
  double *offset);
```

LUA Syntax:

```
offset, rc = mc.mcCntlGetToolOffset(
  number mInst,
  number axisId)
```

Description: Get the tool offset distance for the specified axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	An integer specifying the axis.
offset	The address of a double that receives the offset distance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.
MERROR_INVALID_ARG	**offset** is NULL.

Notes:
None

Usage:

```
int mInst=0;
int toolnum = 5;
double val = 0;
```

```
mcCntlGetToolOffset(mInst, Z_AXIS, &val;); // Get the tool offset distance for tl
```

mcCntlGetUnitsCurrent

C/C++ Syntax:

```
int mcCntlGetUnitsCurrent(
  MINSTANCE mInst,
  int *units);
```

LUA Syntax:

```
units, rc = mc.mcCntlGetUnitsCurrent(
  number mInst)
```

Description: Get the current machine units.

Parameters:

Parameter	Description
mInst	The controller instance.
units	The address of an integer to receive the current machine units. (200 = inches

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**units** is NULL.

Notes:
None.

Usage:

```
// Get the current machine units.
MINSTANCE mInst = 0;
int units;
int rc = mcCntlGetUnitsCurrent(mInst, &units;);
if (rc == MERROR_NOERROR) {
 // Success!
}
```

mcCntlGetUnitsDefault

C/C++ Syntax:

```
int mcCntlGetUnitsDefault(
  MINSTANCE mInst,
  int *units);
```

LUA Syntax:

```
units, rc = mc.mcCntlGetUnitsDefault(
  number mInst)
```

Description: Get the default machine units.

Parameters:

Parameter	Description
mInst	The controller instance.
units	The address of an integer to receive the default machine units. (200 = inches

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**units** is NULL.

Notes:
None.

Usage:

```
// Get the default machine units.
MINSTANCE mInst = 0;
int units;
int rc = mcCntlGetUnitsDefault(mInst, &units;);
if (rc == MERROR_NOERROR) {
 // Success!
}
```

mcCntlGetValue

C/C++ Syntax:

```
int mcCntlGetValue(
  MINSTANCE  mInst,
  int valId,
  int param,
  double *value);
```

LUA Syntax:

```
value, rc = mc.mcCntlGetValue(
  number mInst,
  number valId,
  number param)
```

Description: Get the value of a core variable.

Parameters:

Parameter	Description
mInst	The controller instance.
valId	An integer specifying the core variable (VAL_ prefix in MachAPI.h).
param	An integer specifying the parameter for the core variable
value	The address of a double to receive the value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_IMPLEMENTED	The value specified by **valId** is not implemented.
various	Depending upon the value specified by **valId**.

Notes:
This is for ArtSoft's use. It is a generic interface for retrieving values from the core. Use of this API

function is dicouraged in favor of using one of the clearly named API calls to retrieve values for GUI and plugin programming.

Usage:

```
int mInst = 0;
int motor = 1;
double value = 0;
// Get the motor velocity of motor1.
int rc = mcCntlGetValue(mInst, VAL_MOTOR_VEL, motor, &value;);
```

mcCntlGetVersion

C/C++ Syntax:

```
int mcCntlGetVersion(
  MINSTANCE mInst,
  char *buf,
  size_t bufSize);
```

LUA Syntax:

```
buf, rc = mc.mcCntlGetVersion(
  number mInst)
```

Description: Retrieve the core version string.

Parameters:

Parameter	Description
mInst	The controller instance.
buf	A string buffer to receive the core version string.
bufSize	The length of the string buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**buf** is NULL or **bufSize** is 0.

Notes:
None.

Usage:

```
// Get the core version string.
MINSTANCE mInst = 0;
char ver[80];
int rc = mcCntlGetVersion(mInst, ver, sizeof(ver)):
if (rc == MERROR_NOERROR) {
 printf("The core version is %sn", ver);
}
```

mcCntlGotoZero

C/C++ Syntax:

```
int mcCntlGotoZero(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcCntlGotoZero(
  number mInst)
```

Description: Move the X,Y,A,B,C and then the Z axis to zero of the current fixture offset.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_NOW	The operation could not be completed at this time.

Notes:
This can also be done by sending in a G code move.

Usage:

```
int mInst = 0;
mcCntlGotoZero(mInst); // Move controller instance 0 to x,y,z,a,b,c zero.
```

mcCntlInit

C/C++ Syntax:

```
MINSTANCE mcCntlInit(
  const char *ProfileName,
  int ControllerID);
```

LUA Syntax:

```
N/A
```

Description: Init the instance specified by **ControllerID** with the profile specified by **ProfileName**.

Parameters:

Parameter	Description
ProfileName	Profile that is to be loaded for the controller instance.
ControllerID	The controller instance to start. (must start with zero)

Returns:

Return Code	Description
The controller instance.	0 to 5
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MACHINIT_INVALID_ARG	**ProfileName** is NULL.
MACHINIT_CONTROLLER_INUSE	The controller instance is in use, try the next one.

Notes:
Only GUIs will use this API function. It must be called at the start of the app to initialize at least one controller instance. All other API calls will fail with MERROR_INVALID_INSTANCE The **mInst** parameter was out of range. until an instance is initialized.

Usage:

```
MINSTANCE mInst;
mInst = mcCntlInit("Mach4Mill", mInst);
if (mInst >= 0) {
 // A valid instance has been initialized!
}
```

mcCntlIsInCycle

C/C++ Syntax:

```
int mcCntlIsInCycle(
  MINSTANCE mInst,
  BOOL *cycle);
```

LUA Syntax:

```
cycle, rc = mc.mcCntlIsInCycle(
  number mInst)
```

Description: Used to see if a G code file is running.

Parameters:

Parameter	Description
mInst	The controller instance.
cycle	The address of a BOOL to receive the in cycle state (TRUE, FALSE).

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**cycle** is NULL.

Notes:
This call is useful for showing the user if the control is processing a G code file.

Usage:

```
int mInst = 0;
BOOL InCycle = FALSE;
bool RunningFile = false;

int rc = mcCntlIsInCycle(mInst, &InCycle;); // See if a G code file is running.
if(rc == MERROR_NOERROR && InCycle == TRUE){
    RunningFile = true;
}
```

mcCntlIsStill

C/C++ Syntax:

```
int mcCntlIsStill(
  MINSTANCE mInst,
  BOOL *still);
```

LUA Syntax:

```
still, rc = mcCntlIsStill(
  number mInst)
```

Description: Determine if all controlled axes are still.

Parameters:

Parameter	Description
mInst	The controller instance.
still	The address of a BOOL to receive the still state of the control.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**still** is NULL.

Notes:
This function cannot be used to determine if the control is finished processing G code. For example, the call may return TRUE if the control is in a dwell (G04).

Usage:

```
// See if the control axes are still.
MINSTANCE mInst = 0;
BOOL still;
int rc = mcCntlIsStill(mInst, &still;);
if (rc == MERROR_NOERROR) {
 if (still == TURE) {
  // All axes are still.
 } else {
```

```
  // At least one axis is moving!
 }
}
```

mcCntlLoadGcodeFile

C/C++ Syntax:

```
int mcCntlLoadGcodeFile(
  MINSTANCE mInst,
  const char *FileToLoad);
```

LUA Syntax:

```
rc = mc.mcCntlLoadGcodeFile(
  number mInst,
  stringFileToLoad)
```

Description: Load a G code file.

Parameters:

Parameter	Description
mInst	The controller instance.
FileToLoad	A string buffer containing the Gcode file name.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**FileToLoad** is NULL.
MERROR_FILE_EXCEPTION	Exception while loading file.
MERROR_FILE_EMPTY	No data a in file.
MERROR_FILE_SHARING	Violation sharing file.
MERROR_FILE_INVALID	Not a valid file type.
MERROR_FILE_BADSIZE	Invalid file size (over 1.5GB).
MERROR_FILE_NOT_FOUND	The file cannot be found in the file system.
MERROR_FILE_BADFORMAT	The file had a line end without a new line character. The file may be incomplete.

Notes:
Used to load a file into a controller instance.

Usage:

```
int mInst=0;
char File[128] = "C:SomeGocdefile.tap";
int rc = mcCntlLoadGcodeFile(mInst, &char;); // Load SomeGocdefile.tap file.
if (rc == MERROR_NOERROR) {
 // Success!
}
```

mcCntlLoadGcodeString

C/C++ Syntax:

```
int mcCntlLoadGcodeString(
  MINSTANCE mInst,
  const char *gCode);
```

LUA Syntax:

```
rc = mc.mcCntlLoadGcodeString(
  MINSTANCE mInst,
  const char *gCode);
```

Description: Loads G code from a string.

Parameters:

Parameter	Description
mInst	The controller instance.
gCode	A string buffer containing G code.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**gCode** is NULL.

Notes:
This function will load G code from a string instead of a file. It is primarily used for static test or repeating operations.

Usage:

```
// Load G code from a string.
MINSTANCE mInst = 0;
char *gCode = "%nO1001nG90 G94 G91.1 G40 G49 G17nG20"
int rc = mcCntlLoadGcodeString(mInst, gCode);
```

mcCntlLog

C/C++ Syntax:

```
int mcCntlLog(
 MINSTANCE mInst,
 const char *message,
 const char *file,
 int line);
```

LUA Syntax:

```
rc = mc.mcCntlLog(
 number mInst,
 string message,
 string file,
 number line)
```

Description: Log a message to the core log.

Parameters:

Parameter	Description
mInst	The controller instance.
message	A string buffer containing the message to be logged.
file	A string buffer specifying the source file name.
line	An integer specifying the source line number.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

Logging must be enabled with mcCntlSetLogging() in order for the messages to show in the diagnostic log.

If **file** is NULL, no source file or line information is dispalyed in the log for C/C++ environments. If **line** is -1, no source file or line information is dispalyed in the log for scripting environments.

Usage:

```
// Load G code from a string.
MINSTANCE mInst = 0;
BOOL test = TRUE;
int rc;
if (test == TRUE) {
 // File and line info in the log.
 mcCntlLog(mInst, "test == TRUE!", __FILE__, __LINE__);
} else {
 // No file and line info in the log.
 mcCntlLog(mInst, "test == FALSE!", NULL, 0);
}
```

mcCntlMachineStateClear

C/C++ Syntax:

```
int mcCntlMachineStateClear(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcCntlMachineStateClear(
  number mInst)
```

Description: Clears the machine control state stack.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
This function is used to clear a stack of control states and is primarily used in a scripting environment. Control states are pushed on a stack with mcCntlMachineStatePush().

Usage:

```
-- LUA example
function test()
 local inst = mc.mcGetInstance();
 -- push control state to the stack.
 mc.mcCntlMachineStatePush(inst);
 -- put machine in G20, and G90 mode.
 mc.mcCntlGcodeExecuteWait(inst, "G20nG90");
 -- push control state to the stack.
 mc.mcCntlMachineStatePush(inst);
 -- put machine in G21, and G91 mode.
 mc.mcCntlGcodeExecuteWait(inst, "G21nG91");
 -- clear the machine state stack and leave the machine in G21 and G91 modes.
 mc.mcCntlMachineStateClear(inst);
end
```

mcCntlMachineStatePop

C/C++ Syntax:

```
int mcCntlMachineStatePop(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcCntlMachineStatePop(
  number mInst)
```

Description: Pops a machine control state off the stack.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
This function is used to return the control to a state previously saved on the state stack and is primarily used in a scripting environment. Control states are pushed (saved) on a stack with mcCntlMachineStatePush().

Usage:

```
-- LUA example
function test()
-- LUA example
function test()
 local inst = mc.mcGetInstance();
 -- push control state to the stack saving original modes.
 mc.mcCntlMachineStatePush(inst);
 -- put machine in G20, and G90 mode.
 mc.mcCntlGcodeExecuteWait(inst, "G20nG90");
 -- push control state to the stack.
 mc.mcCntlMachineStatePush(inst);
 -- put machine in G21, and G91 mode.
 mc.mcCntlGcodeExecuteWait(inst, "G21nG91");
```

```
 -- restore the machine state stack to G20 and G90 modes.
 mc.mcCntlMachineStatePop(inst);
 -- restore the machine state stack to original modes.
 mc.mcCntlMachineStatePop(inst);
end
```

mcCntlMachineStatePush

C/C++ Syntax:

```
int mcCntlMachineStatePush(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcCntlMachineStatePush(
  number mInst)
```

Description: Pushes the machine control state on the stack.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
This function is used to save the control state to a stack of states and is primarily used in a scripting environment.

Usage:

```
-- LUA example
function test()
 local inst = mc.mcGetInstance();
 -- push control state to the stack saving original modes.
 mc.mcCntlMachineStatePush(inst);
 -- put machine in G20, and G90 mode.
 mc.mcCntlGcodeExecuteWait(inst, "G20nG90");
 -- push control state to the stack.
 mc.mcCntlMachineStatePush(inst);
 -- put machine in G21, and G91 mode.
 mc.mcCntlGcodeExecuteWait(inst, "G21nG91");
 -- restore the machine state stack to G20 and G90 modes.
 mc.mcCntlMachineStatePop(inst);
 -- restore the machine state stack to original modes.
```

```
 mc.mcCntlMachineStatePop(inst);
end
```

mcCntlMdiExecute

C/C++ Syntax:

```
int mcCntlMdiExecute(
  MINSTANCE mInst,
  const char *commands);
```

LUA Syntax:

```
rc = mc.mcCntlMdiExecute(
  number mInst,
  string commands)
```

Description: Used to execute G code commands that are not part of the main G code execution.

Parameters:

Parameter	Description
mInst	The controller instance.
commands	A string buffer with the line or lines of Gcode.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**commands** is NULL.

Notes:
Used to run commands like MDI or script commands. The function returns immediately without waiting on the G code to proocess. It is an error to call this function multiple times for each block of G code. Multiple blocks should be concatenated and separated with new line characters to form a "unit of work".

Usage:

```
MINSTANCE mInst = 0;
int rc;
// Move the machine back to xy then z zero.
// Correct example.
rc = mcCntlMdiExecute(mInst, "G00 G90 X0.0 Y0.0\nZ0.0");
```

```
// Incorrect example.
rc = mcCntlMdiExecute(mInst, "G00 G90 X0.0 Y0.0");
rc = mcCntlMdiExecute(mInst, "Z0.0");
```

mcCntlProbeFileClose

C/C++ Syntax:

```
int mcCntlProbeFileClose(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcCntlProbeFileClose(
  number mInst)
```

Description: Close the probe data file.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Closes a probe data file that was previously opened with mcCntlProbeFileOpen(). This function is primarily used in a scripting environment.

Usage:

```
-- probe file close LUA macro example.
function m112()
 local inst=mc.mcGetInstance();
 local rc = mc.mcCntlProbeFileClose(inst);
end

if (mc.mcInEditor() == 1) then
 m112()
end
```

mcCntlProbeFileOpen

C/C++ Syntax:

```
int mcCntlProbeFileOpen(
  MINSTANCE mInst,
  const char *fileName,
  const char *format,
  BOOL overWrite);
```

LUA Syntax:

```
rc = mc.mcCntlProbeFileOpen(
  number mInst,
  string fileName,
  string format,
  number overWrite);
```

Description: Open a probe data collection file.

Parameters:

Parameter	Description
mInst	The controller instance.
fileName	A string buffer containing the full path of the probe data file to be opened.
format	A string buffer containing the format description for the probe data file.
overWrite	A BOOL specifying if the file should be over written if it exists..

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

Opens a probe data collection file and specifies the format in which the data is saved. This function is primarily used in a scripting environment.

The **format** parameter uses subtitutions. Numbers are formatted with the C/C++ printf style notation. Axis values are substituted with AXIS_X, AXIS_Y, AXIS_Z, AXIS_A, AXIS_B, and AXIS_C.

Usage:

```
-- probe file open LUA macro example.
function m111()
 local inst = mc.mcGetInstance();
 local rc = mc.mcCntlProbeFileOpen(inst, 'probetest.csv', '%.4AXIS_X, %.4AXIS_Y,
end

if (mc.mcInEditor() == 1) then
 m111()
end
```

mcCntlReset

C/C++ Syntax:

```
int mcCntlReset(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcCntlReset(
  number mInst)
```

Description: Reset the controller instance to a known state.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_NOW	The operation could not be completed at this time.

Notes:
A MSG_RESET_CNTL notification is sent to the GUI and plugins.

Usage:

```
MINSTANCE mInst = 0;
mcCntlReset(mInst); // Reset controller instance 0.
```

mcCntlRewindFile

C/C++ Syntax:

```
int mcCntlRewindFile(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcCntlRewindFile(
  number mInst)
```

Description: Rewind the G code file.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_NOW	The G code file is running and not allowing a rewind.

Notes:
A MSG_REWIND_FILE notification is sent to the GUI and plugins.

Usage:

```
MINSTANCE mInst = 0;
mcCntlRewindFile(mInst); // Rewind controller 0's Gcode file.
```

mcCntlSetBlockDelete

C/C++ Syntax:

```
int mcCntlSetBlockDelete(
  MINSTANCE mInst,
  int deleteID,
  BOOL enabled);
```

LUA Syntax:

```
rc = mc.mcCntlSetBlockDelete(
  number mInst,
  number deleteID,
  number enabled)
```

Description: Enable or Disable a block delete level.

Parameters:

Parameter	Description
mInst	The controller instance.
deleteId	The block delete index level.
enabled	A BOOL specifying if the lock delete level is enabled. (TRUE/FALSE)

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**deleteID** is out of range.

Notes:
There are currently 10 block delete levels (0-9). If any block delete level is enabled, the OSIG_BLOCK_DELETE output signal is asserted.

Usage:

```
MINSTANCE mInst = 0;
// Enable block delete level 0.
mcCntlSetBlockDelete(mInst, 0, TRUE);
```

mcCntlSetCoolantDelay

C/C++ Syntax:

```
int mcCntlSetCoolantDelay(
  MINSTANCE mInst,
  double secs);
```

LUA Syntax:

```
rc = mc.mcCntlSetCoolantDelay(
  number mInst,
  number secs);
```

Description: Set the coolant delay in seconds.

Parameters:

Parameter	Description
mInst	The controller instance.
seconds	A double specifying the delay in seconds.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Seconds can be a decimal value. e.g. .5 for half of a second.

Usage:

```
// Set 3/4 second coolant delay.
MINSTANCE mInst = 0;
int rc = mcCntlSetCoolantDelay(mInst, .750);
```

mcCntlSetDiaMode

C/C++ Syntax:

```
int mcCntlSetDiaMode(
  MINSTANCE mInst,
  BOOL enable);
```

LUA Syntax:

```
rc = mc.mcCntlSetDiaMode(
  number mInst,
  number enable);
```

Description:

Parameters:

Parameter	Description
mInst	The controller instance.
enable	A BOOL specifying the siameter mode state. (TRUE/FALSE)

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

This function is only valid when the control is in lathe mode.

Usage:

```
// Enable diamter mode
MINSTANCE mInst = 0;
int rc = mcCntlSetDiaMode(mInst, TRUE);
```

mcCntlSetFRO

C/C++ Syntax:

```
int mcCntlSetFRO(
  MINSTANCE mInst,
  double percent);
```

LUA Syntax:

```
rc = mc.mcCntlSetFRO(
  number mInst,
  number percent);
```

Description: Set the feed rate override by the percentage of the coded feed rate.

Parameters:

Parameter	Description
mInst	The controller instance.
percent	A double specifying the feed rate override percent.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

Usage:

```
// Set feed rate override to 150%
MINSTANCE mInst = 0;
int rc = mcCntlSetFRO(mInst, 150.0);
```

mcCntlSetGcodeLineNbr

C/C++ Syntax:

```
int mcCntlSetGcodeLineNbr(
  MINSTANCE mInst,
  double line);
```

Description: Set the Gcode line number to run / start.

Parameters:

Parameter	Description
mInst	The controller instance.
line	Sets the current line in the Gcode file.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
This will not update the running state engine so great care should be used when setting the line.
Do not set the line when processing G code.
Calling this function sends MSG_GCODE_LINE to the GUI and plugins.
The parameter **line** is base 1.

Usage:

```
int mInst=0;
mcCntlSetGcodeLineNbr(mInst, 10); // Set the Gcode file to be at line 10.
```

mcCntlSetLastError

C/C++ Syntax:

```
int mcCntlSetLastError(
  MINSTANCE mInst,
  const char *emsg);
```

LUA Syntax:

```
rc = mc.mcCntlSetLastError(
  number mInst,
  string emsg)
```

Description: Sets the last error in the error stack.

Parameters:

Parameter	Description
mInst	The controller instance.
emsg	A string buffer with the error message.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
The error sent may not be seen by the user if the caller of mcGetLastError() is not showing the last errors.

Usage:

```
MINSTANCE mInst = 0;
char *erbuf = "Tool no longer in the spindle";
// Send an error messge for controller 0.
int rc = mcCntlSetLastError(mInst, erbuf);
```

mcCntlSetLogging

C/C++ Syntax:

```
int mcCntlSetLogging(
  MINSTANCE mInst,
  BOOL enable);
```

LUA Syntax:

```
rc = mc.mcCntlSetLogging(
  number mInst,
  number enable)
```

Description: Enable or disable the log facility.

Parameters:

Parameter	Description
mInst	The controller instance.
enable	A BOOL specifying the state of the logging facility.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Enable logging.
MINSTANCE mInst = 0;
int rc = mcCntlSetLogging(mInst, enable);
```

mcCntlSetMistDelay

C/C++ Syntax:

```
int mcCntlSetMistDelay(
  MINSTANCE mInst,
  double secs);
```

LUA Syntax:

```
rc = mc.mcCntlSetMistDelay(
  number mInst,
  number secs);
```

Description: Set the delay for the mist coolant in seconds.

Parameters:

Parameter	Description
mInst	The controller instance.
secs	A double specifying the delay in seconds.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Decimal values are allowed. e.g. .5 is 1/2 of a second.

Usage:

```
// Set the mist delay to 3/4 of a second.
MINSTANCE mInst = 0;
int rc = mcCntlSetMistDelay(mInst, .750);
```

mcCntlSetMode

C/C++ Syntax:

```
int mcCntlSetMode(
  MINSTANCE mInst,
  double mode);
```

LUA Syntax:

```
rc = mc.mcCntlSetMode(
  number mInst,
  number mode)
```

Description: Set the interpreter mode of the controller instance.

Parameters:

Parameter	Description
mInst	The controller instance.
mode	the interpreter mode of the controller. (MC_MODE_MILL, MC_MODE_LATHE_DIA, MC_MODE_LATHE_RAD, MC_MODE_TANGENTIAL)

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
The core needs to be restarted if the mode is changed.

Usage:

```
// Set the interpreter mode to Lathe Diameter.
MINSTANCE mInst = 0;
int rc = mcCntlSetMode(mInst, MC_MODE_LATHE_DIA);
```

mcCntlSetOptionalStop

C/C++ Syntax:

```
int mcCntlSetOptionalStop(
  MINSTANCE mInst,
  BOOL enable);
```

LUA Syntax:

```
rc = mc.mcCntlSetOptionalStop(
  number mInst
  number enable)
```

Description: Set the state of optional stop.

Parameters:

Parameter	Description
mInst	The controller instance.
enable	A BOOL specifying the state of optional stop. (TRUE/FALSE)

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
The output signal OSIG_OPT_STOP reflects the state of optional stop.

Usage:

```
// Enable optional stop.
MINSTANCE mInst = 0;
int rc = mcCntlSetOptionalStop(mInst, TRUE);
```

mcCntlSetPoundVar

C/C++ Syntax:

```
mcCntlSetPoundVar(
  MINSTANCE mInst,
  int param,
  double value);
```

LUA Syntax:

```
mcCntlSetPoundVar(
  number mInst,
  number param,
  number value)
```

Description: Set the value of a system variable (pound variable).

Parameters:

Parameter	Description
mInst	The controller instance.
param	system variable index.
value	The value to apply to the system variable (pound variable).

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**param** is out of range.

Notes:
This function should only be used when G code is not processing.

Usage:

```
MINSTANCE mInst = 0;
int PoundVar = 50;
dobule Value=21;
// Set the value of #50 to 21.
int rc = mcCntlSetPoundVar(mInst, PoundVar, Value);
```

mcCntlSetRRO

C/C++ Syntax:

```
int mcCntlSetRRO(
  MINSTANCE mInst,
  double percent);
```

LUA Syntax:

```
rc = mc.mcCntlSetRRO(
  number mInst,
  number percent);
```

Description: Set the rapid rate override by the percentage of maxiumum velocity.

Parameters:

Parameter	Description
mInst	The controller instance.
percent	A double specifying the rapid rate override percentage.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
The maximum velocity is capped by the motor tuning. Therefore the percentage should be less than 100%.

Usage:

```
// Set feed rapid override to 50%
MINSTANCE mInst = 0;
int rc = mcCntlSetRRO(mInst, 50.0);
```

mcCntlSetResetCodes

C/C++ Syntax:

```
int mcCntlSetResetCodes(
  MINSTANCE mInst,
  const char *resetCodes);
```

LUA Syntax:

```
rc = mc.mcCntlSetResetCodes(
  number mInst,
  string resetCodes)
```

Description: Set the G code that is used when mcCntlReset() is called.

Parameters:

Parameter	Description
mInst	The controller instance.
resetCodes	A string buffer containing the G code.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Each block needs to be separated with new line characters (\n).

Usage:

```
// Set the G code used to reset the controller instance 0.
MINSTANCE mInst = 0;
int rc = mcCntlSetResetCodes(mInst, "G40 G20 G90 G52 X0 Y0 Z0\nG92.1 G69");
```

mcCntlSetSingleBlock

C/C++ Syntax:

```
int mcCntlSetSingleBlock(
  MINSTANCE mInst,
  BOOL enable);
```

Description: Set the state of single block mode.

Parameters:

Parameter	Description
mInst	The controller instance.
enable	A BOOL specifying the state of single block mode. (TRUE/FALSE).

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Single block runs one line of G code at a time and is most often used to debug G code programs.

Usage:

```
// Enable single block mode.
MINSTANCE mInst = 0;
mcCntlSetSingleBlock(mInst, TRUE);
```

mcCntlSetStats

C/C++ Syntax:

```
int mcCntlSetStats(
  MINSTANCE mInst,
  mstats_t *stats);
```

LUA Syntax:

```
N/A
```

Description: Set/reset the machine statistics.

Parameters: (stats, The address of a mstats_t struct with which to set the control statistics.

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**stats** is NULL.

Notes:

Usage:

```
//
MINSTANCE mInst = 0;
```

mcCntlSetValue

C/C++ Syntax:

```
int mcCntlSetValue(
  MINSTANCE  mInst,
  int valId,
  int param,
  double value);
```

LUA Syntax:

```
rc = mc.mcCntlSetValue(
  number mInst,
  number valId,
  number param,
  number value)
```

Description: Set the value of a control variable.

Parameters:

Parameter	Description
mInst	The controller instance.
valId	The control variable to set (VAL_ prefix in MachAPI.h).
param	The parameter for the control value (or 0).
value	The new value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_IMPLEMENTED	The value specified by **valId** is not implemented.
Various	Depending upon **valId**.

Notes:
This function is used by Artsoft to add control values to the core on a per OEM basis. Use of this function is discouraged in favor of using the more clearly named API functions for GUI and plugin

programming.

Usage:

```
MINSTANCE mInst = 0;
int motor = 1;
value=1000.0;
// Set the Velocity of the motor to 1000.0 .
in rc = mcCntlSetValue(mInst, VAL_MOTOR_VEL, motor, value);
```

mcCntlStartMotionDev

C/C++ Syntax:

```
int mcCntlStartMotionDev(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcCntlStartMotionDev(
  number mInst)
```

Description: Starts the selected motion device.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Used to start the selected motion device. This is normally a function used by a GUI where a motion device selection dialog is presented to the user.

Usage:

```
// Start the selected motion device.
MINSTANCE mInst = 0;
int rc = mcCntlStartMotionDev(mInst);
```

mcCntlStopMotionDev

C/C++ Syntax:

```
int mcCntlStopMotionDev(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcCntlStopMotionDev(
  number mInst);
```

Description: Stop the selected motion device.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Used to stop the selected motion device. This is normally a function used by a GUI where a motion device selection dialog is presented to the user.

Usage:

```
// Stop the selected motion device.
MINSTANCE mInst = 0;
int rc = mcCntlStopMotionDev(mInst);
```

mcCntlToolChangeManual

C/C++ Syntax:

```
int mcCntlToolChangeManual(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcCntlToolChangeManual(
  number mInst)
```

Description: A convenience function used in M6 tool change macros. It will prompt the user to change the tool and wait for a cycle start.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_NOW	The operation could not be completed at this time.

Notes:
This function is primarily used in a scripting environment.

Usage:

```
-- LUA example
local inst = mc.mcGetInstance();
local rc = mc.mcCntlToolChangeManual(inst);
```

mcCntlWaitOnCycleStart

C/C++ Syntax:

```
int mcCntlWaitOnCycleStart(
  MINSTANCE mInst,
  const char *msg,
  int timeOutMs);
```

LUA Syntax:

```
rc = mc.mcCntlWaitOnCycleStart(
  number mInst,
  string msg,
  number timeOutMs);
```

Description: A convenience function used in macros. It will prompt the user with the given message and optionally use a timeout.

Parameters:

Parameter	Description
mInst	The controller instance.
msg	A string buffer for the prompt message.
timeOutMs	A timeout value in milliseconds.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_NOW	The operation could not be completed at this time.

Notes:
This function is primarily used in a scripting environment.

Usage:

```
-- LUA example
local inst = mc.mcGetInstance();
local rc = mc.mcCntlWaitOnCycleStart(inst, "Please press Cycle Start.", 1000);
```

mcDeviceGetHandle

C/C++ Syntax:

```
int mcDeviceGetHandle(
  MINSTANCE mInst,
  int devid,
  HMCDEV *hDev);
```

LUA Syntax:

```
hDev, rc = mc.mcDeviceGetHandle(
  number mInst,
  number devid)
```

Description: Retrieve the hDev handle specified by devid.

Parameters:

Parameter	Description
mInst	The controller instance.
devid	The device ID as it was registered in the core.
hDev	The address of a HMCDEV handle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_DEVICE_NOT_FOUND	No device with the ID specified by **devid**was found.

Notes:
None.

Usage:

```
int mInst = 0;
int devid = 0;
HMCDEV hDev;
//See if we can find a device with an I of zero
int rc = mcDeviceGetHandle(mInst, devid, &hDev;);
```

```
if (rc == MERROR_NOERROR) {
 // We found it!
}
```

mcDeviceGetInfo

C/C++ Syntax:

```
int mcDeviceGetInfo(
  HMCDEV hDev,
  char *nameBuf,
  size_t nameBuflen,
  char *descBuf,
  size_t descBuflen,
  int *type,
  int *id);
```

LUA Syntax:

```
nameBuf, descBuf, type, id, rc = mc.mcDeviceGetInfo(HMCDEV hDev}
```

Description: Return infromation about a device.

Parameters:

Parameter	Description
hDev	The handle for the device.
nameBuf	The address of a string buffer to receive the device name.
nameBuflen	The nameBuf buffer length.
descBuf	The address of a string buffer to receive the device description.
descBuflen	The descBuf buffer length.
type	The address of an integer to receive the device type.
id	The address of an integer to receive the device ID.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The **hDev** parameter was 0 or no device with that handle exists.

Notes:
None.

Usage:

```
int mInst=0;
HMCSIG hDev = 0;
devinfo_t devinf;
char nameBuf[80];
char descBuf[80];
int type = 0;
int id = 0;
//Look for all the IO devices and get their devinfo_t struct
while (mcDeviceGetNextHandle(mInst, DEV_TYPE_IO, hDev, &hDev;) == MERROR_NOERROR)
 if (hSig != 0) {
  mcDeviceGetInfo(hDev, nameBuf, sizeof(nameBuf),
    descBuf, sizeof(descBuf), &type;, &id;);
// do something with the device info.
 } else {
  break;
 }
}
```

mcDeviceGetInfoStruct

C/C++ Syntax:

```
int mcDeviceGetInfoStruct(
  HMCSIG hDev,
  devinfo_t *devinf);
```

LUA Syntax:

```
N/A
```

Description: Return infromation about a device.

Parameters:

Parameter	Description
hDev	The handle for the device.
devinf	The address of a devinfo struct to receive the device information.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The **hDev** parameter was 0.

Notes:

```
struct devinfo {
 char devName[80];
 char devDesc[80];
 int  devType;
 int  devId;
};
typedef struct devinfo devinfo_t;
```

Usage:

```
int mInst=0;
HMCSIG hDev = 0;
devinfo_t devinf;

//Look for all the IO devices and get their devinfo_t struct
```

```
while (mcDeviceGetNextHandle(mInst, DEV_TYPE_IO, hDev, &hDev;) == MERROR_NOERROR)
 if (hSig != 0) {
  mcDeviceGetInfoStruct(hDev, devinf);
// Do something with the device info.
 } else {
  break;
 }
}
```

mcDeviceGetNextHandle

C/C++ Syntax:

```
int mcDeviceGetNextHandle(
  MINSTANCE mInst,
  int devtype,
  HMCDEV startDev,
  HMCDEV *hDev);
```

LUA Syntax:

```
hDev, rc = mc.mcDeviceGetNextHandle(
  number mInst,
  number devtype,
  number startDev)
```

Description: Provides a means of looping through the available devices.

Parameters:

Parameter	Description
mInst	The controller instance.
devtype	Device type DEV_TYPE_NONE, DEV_TYPE_MOTION, DEV_TYPE_IO, DEV_TYPE_SCRIPT, DEV_TYPE_CONFIG, DEV_TYPE_DIAG.
startDev	A device handle defining the device from which to start.
hDev	A pointer to a HMCDEV to receive the next device handle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NODATA	No more device handles are available.

Notes:
None.

Usage:

```
int mInst=0;
HMCDEV hDev;
//Look for all the IO devices
while (mcDeviceGetNextHandle(mInst, DEV_TYPE_IO, hDev, &hDev;) == MERROR_NOERROR)
 if (hSig != 0) {
  // Do something with hDev.
 } else {
  break;
 }
}
```

mcDeviceRegister

C/C++ Syntax:

```
mcDeviceRegister(
 MINSTANCE mInst,
 HMCPLUG plugid,
 const char *DeviceName,
 const char *DeviceDesc,
 int Type,
 HMCDEV *hDev);
```

LUA Syntax:

```
N/A
```

Description: Register the device in the core.

Parameters:

Parameter	Description
mInst	The controller instance.
plugid	The ID of the plugin passed by mcPluginInit.
DeviceName	Name the device.
DeviceDesc	Description of the device.
Type	DEV_TYPE_NONE, DEV_TYPE_MOTION, DEV_TYPE_IO, DEV_TYPE_SCRIPT, DEV_TYPE_CONFIG, DEV_TYPE_DIAG, DEV_TYPE_REG, DEV_TYPE_PROBE2, DEV_TYPE_THREAD2, DEV_TYPE_RTAP, DEV_TYPE_RTAP2, DEV_TYPE_PROBE, DEV_TYPE_THREAD
hDev	The address of a HMCDEV to receive the returned device handle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

MERROR_PLUGIN_NOT_FOUND	The plugin referenced by **pluginid** was not found.
MERROR_NOT_CREATED	The device was not registered!

Notes:
Registers a device into the core and defines its capabilities.

Usage:

```
simControl::simControl(MINSTANCE mInst, HMCPLUG id)
{
 m_cid = mInst;
 m_id = id;
 m_timer = new simTimer(this);
 m_cycletime = .001;

 mcDeviceRegister(m_cid, m_id, "Sim0", "Simulation Device", DEV_TYPE_MOTION | DE

 mcRegisterIo(m_hDev, "Input0", "Input0", IO_TYPE_INPUT, &m;_Input0);
 mcRegisterIo(m_hDev, "Output0", "Output0", IO_TYPE_OUTPUT, &m;_Output0);

 if (!m_timer->Start(10, wxTIMER_ONE_SHOT)) {
  wxMessageBox(wxT("Timer could not start!"), wxT("Timer"));
 }
}
```

mcFileHoldAquire

C/C++ Syntax:

```
int mcFileHoldAquire(
  MINSTANCE mInst,
  const char *reason,
  int JogAxisBits);
```

LUA Syntax:

```
rc = mc.mcFileHoldAquire(
  number mInst,
  string reason,
  number JogAxisBits)
```

Description: Used to hold G code processing.

Parameters:

Parameter	Description
mInst	The controller instance.
reason	A string buffer specifying the reason for the file hold.
JogAxisBits	A bitmask of bits specifying the axes that are allowed to jog when in then file hold state.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_STATE	The control is not in a state where file hold is allowed.

Notes:
This function is primarily used in a scripting environment.

Usage:

```
-- LUA example
local inst = mc.mcGetInstance();
local rc = mc.mcFileHoldAquire(inst, "My hold reason", 0);
```

mcFileHoldReason

C/C++ Syntax:

```
int mcFileHoldReason(
  MINSTANCE mInst,
  char *buf,
  long bufSize);
```

LUA Syntax:

```
reason, rc = mc.mcFileHoldReason(
  number mInst)
```

Description: Returns a string clarifying why the file processing was held.

Parameters:

Parameter	Description
mInst	The controller instance.
buf	A string buffer that receives the reason the file was held.
bufSize	A long specifying the length of the string buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_STATE	The control is not in the file hold state.
MERROR_INVALIS_ARG	**buf** is NULL or **bufSize** is less than or equal to 0.

Notes:
This function is primarily used in a scripting environment.

Usage:

```
-- LUA example
local inst = mc.mcGetInstance();
local rc = mc.mcFileHoldAquire(inst, "My hold reason", 0);
loacl reason
```

```
reason, rc = mc.mcFileHoldReason(inst);
```

mcFileHoldRelease

C/C++ Syntax:

```
int mcFileHoldRelease(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcFileHoldRelease(
  number mInst)
```

Description: Used to exit a file hold state.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_STATE	The control was not in the file hold state.

Notes:
This function is primarily used in a scripting environment.

Usage:

```
-- LUA example
local inst = mc.mcGetInstance();
local rc = mc.mcFileHoldAquire(inst, "My hold reason", 0);
loacl reason
reason, rc = mc.mcFileHoldReason(inst);
-- Do some script magic...
rc = mc.mcFileHoldRelease(inst);
-- G code processing resumes.
```

mcFixtureLoadFile

C/C++ Syntax:

```
int mcFixtureLoadFile(
  MINSTANCE mInst,
  const char *FileToLoad);
```

LUA Syntax:

```
rc = mc.mcFixtureLoadFile(
  number mInst,
  string FileToLoad);
```

Description: Used to load a fixture table into the core.

Parameters:

Parameter	Description
mInst	The controller instance.
FileToLoad	A string buffer specifying the fixture table file.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_FILE_EMPTY	The file did not contain any fixture table data.

Notes:
This function is primarily used by the GUI.

Usage:

```
// Load a fixture table file.
MINSTANCE mInst = 0;
int rc = mcFixtureLoadFile(mInst, "MyFixtureTable.dat");
```

mcFixtureSaveFile

C/C++ Syntax:

```
int mcFixtureSaveFile(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcFixtureSaveFile(
  number mInst),
```

Description: Used to save a previously loaded and modified fixture table file.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_FILE_INVALID	No fixture table file was previously loaded.

Notes:
This function is primarily used by the GUI.

Usage:

```
// Save the fixture table file.
MINSTANCE mInst = 0;
int rc = mcFixtureSaveFile(mInst);
```

mcGuiGetWindowHandle

C/C++ Syntax:

```
int mcGuiGetWindowHandle(
  MINSTANCE mInst,
  void **handle);
```

LUA Syntax:

```
N/A
```

Description: Used by the core or plugins to retriev the GUI's main window handle.

Parameters:

Parameter	Description
mInst	The controller instance.
handle	The address of a void pointer to receive the GUI's main window handle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
If the returned handle is NULL, then no handle was set by the GUI.

Usage:

```
//
MINSTANCE mInst = 0;
void *guiHandle;
rc = mcGuiGetWindowHandle(mInst, &guiHandle;);
```

mcGuiSetCallback

C/C++ Syntax:

```
int mcGuiSetCallback(
 MINSTANCE mInst,
 void *fp);
```

LUA Syntax:

```
N/A
```

Description: Sets a call back function that is used to pass messages from the core to the GUI.

Parameters:

Parameter	Description
mInst	The controller instance.
fp	Function pointer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
If this call back is not set, then the GUI has no means of receiving event based messages.

Usage:

```
/* In the GUI code */
MCP_API int MCP_APIENTRY mcGUIMsg(MINSTANCE mInst, long msg, long wparam, long lp
{
 // Process messages...
 return(MERROR_NOERROR);
}

MINSTANCE mInst = 0;
mcGuiSetCallback(mInst, &mcGUIMsg;);
```

mcGuiSetFocus

C/C++ Syntax:

```
int mcGuiSetFocus(
  MINSTANCE mInst,
  BOOL focus);
```

LUA Syntax:

```
N/A
```

Description: Causes the GUI to gain or lose focus.

Parameters:

Parameter	Description
mInst	The controller instance.
focus	A BOOL specifying the desired GUI focus state. (TRUE/FALSE)

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
The GUI will receive a MSG_GUI_FOCUS message. It is up to the GUI to implement this functionality.

Usage:

```
// Set focus to the GUI's main window
MINSTANCE mInst = 0;
int rc = mcGuiSetFocus(mInst, TRUE);
```

mcGuiSetWindowHandle

C/C++ Syntax:

```
int mcGuiSetWindowHandle(
  MINSTANCE mInst,
  void *handle);
```

LUA Syntax:

```
N/A
```

Description: Informs the core of the GUI's main window handle.

Parameters:

Parameter	Description
mInst	The controller instance.
handle	The GUI's main window handle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
It is up to the GUI to implement this functionality. The handle should be the HWND handle on the Windows platform and the wxWindow handle on all other platforms.

Usage:

```
// Set the GUI main window handle.
MINSTANCE mInst = 0;
 mcGuiSetWindowHandle(0, this);
```

mcIoGetHandle

C/C++ Syntax:

```
mcIoGetHandle(
 MINSTANCE mInst,
 const char *path,
 HMCIO *hIo);
```

LUA Syntax:

```
hIo, rc = mc.mcIoGetHandle(
  number mInst,
  string path)
```

Description: Retrieve the IO handle given a named path.

Parameters:

Parameter	Description
mInst	The controller instance.
path	A string representing the named path. e.g. devname/ioname
hIo	The address of a HMCIO handle

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_IO_NOT_FOUND	No IO handle available for the given path.

Notes:
None.

Usage:

```
HMCIO hIo;
char *path = "Sim0/Input0";

mcIoGetHandle(m_cid, path, &hIo;);
```

mcIoGetInfoStruct

C/C++ Syntax:

```
mcIoGetInfoStruct(
 HMCIO hIo,
 ioinfo_t *ioinf);
```

LUA Syntax:

```
N/A
```

Description: Retrieve the current state of the IO handle.

Parameters:

Parameter	Description
hIo	The IO handle.
ioinf	The address of an ioinf struct to receive the IO information.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The **hIo** handle is 0 or invalid, or the **ioinf** parameter is NULL

Notes:
The hIo parameter **MUST** be a valid IO handle. Otherwise, the function will crash.

```
struct ioinfo {
 char ioName[80];
 char ioDesc[80];
 int  ioType;
 HMCDEV ioDev;
 HMCSIG ioMappedSignals[MAX_MAPPED_SIGNAL];
 void *ioUserData;
 int ioInput;
};
typedef struct ioinfo ioinfo_t;
```

Usage:

```
HMCDEV hDev = 0;
devinfo_t devinf;
int rc;

// Get all IO information from every registered device.
while (mcDeviceGetNextHandle(0, DEV_TYPE_IO, hDev, &hDev;) == MERROR_NOERROR) {
 if (mcDeviceGetInfoStruct(hDev, &devinf;) == MERROR_NOERROR) {
  HMCIO hIo = 0;
  while (mcIoGetNextHandle(hDev, hIo, &hIo;) == MERROR_NOERROR) {
   ioinfo_t ioinf;
   rc = mcIoGetInfoStruct(hIo, &ioinf;);
   if (rc ==

MERROR_NOERROR No Error.) {
    // IO information successfuly retrieved.
   }
  }
 }
}
```

mcIoGetNextHandle

C/C++ Syntax:

```
mcIoGetNextHandle(
 HMCDEV hDev,
 HMCIO startIo
 HMCIO *hIo)
```

LUA Syntax:

```
hIo, rc = mc.mcIoGetNextHandle(
 number hDev,
 number startIo)
```

Description: Provides a means to loop though the registered IO of a registered device.

Parameters:

Parameter	Description
hDev	The device handle.
startIo	The starting IO handle. (0 to begin)
hIo	The address to a HMCIO handle to receive the next IO handle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The **hDev** paramete was 0 or invalid.
MERROR_NODATA	No IO handle available.

Notes:
The hSig parameter **MUST** be a valid signal handle. Otherwise, the function will crash.
Useful in a GUI where retrieving information to display to the user is needed.

Usage:

```
HMCDEV hDev = 0;
devinfo_t devinf;
int rc;

// Get all IO information from every registered device.
```

```
while (mcDeviceGetNextHandle(0, DEV_TYPE_IO, hDev, &hDev;) == MERROR_NOERROR) {
 if (mcDeviceGetInfoStruct(hDev, &devinf;) == MERROR_NOERROR) {
  HMCIO hIo = 0;
  while (mcIoGetNextHandle(hDev, hIo, &hIo;) == MERROR_NOERROR) {
   ioinfo_t ioinf;
   rc = mcIoGetInfoStruct(hIo, &ioinf;);
   if (rc == MERROR_NOERROR) {
    // IO information successfuly retrieved.
   }
  }
 }
}
```

mcIoGetState

C/C++ Syntax:

```
mcIoGetState(
 HMCIO hIo,
 BOOL *state);
```

LUA Syntax:

```
state, rc = mcIoGetState(
 number hIo)
```

Description: Retrieve the current state of the IO handle.

Parameters:

Parameter	Description
hIo	The IO handle.
state	The address of a BOOL to receive the state status.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The **hIo** parameter was 0 or invalid.

Notes:
The hIo parameter **MUST** be a valid IO handle. Otherwise, the function will crash.

Usage:

```
simControl::simControl(MINSTANCE mInst, HMCPLUG id)
{
 m_cid = mInst;
 m_id = id;
 m_timer = new simTimer(this);
 m_cycletime = .001;
 bool oldstate = false;
 bool newstate = false;

 mcDeviceRegister(m_cid, m_id, "Sim0", "Simulation Device", DEV_TYPE_MOTION | DEV

 mcIoRegister(m_hDev, "Input0", "Input0", IO_TYPE_INPUT, &m;_Input0);
```

```
 mcIoGetState(m_Input0, &oldstate;);
 newstate = simGetIO(0);
 if (newstate != oldstate) {
  mcIoSetState(m_Input0, newstate);
 }
}
```

mcIoGetType

C/C++ Syntax:

```
mcIoGetType(
 HMCIO hIo,
 int *type);
```

LUA Syntax:

```
type, rc = mcIoGetType(
 number hIo)
```

Description: Retrieve the type (input or output) the IO object.

Parameters:

Parameter	Description
hIo	The IO handle.
type	The address of an integer to receive the type information. (IO_TYPE_INPUT or IO_TYPE_OUTPUT)

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The **hIo** parameter was 0 or invalid.

Notes:
The hIo parameter **MUST** be a valid IO handle. Otherwise, the function will crash.

Usage:

```
simControl::simControl(MINSTANCE mInst, HMCPLUG id)
{
 m_cid = mInst;
 m_id = id;
 m_timer = new simTimer(this);
 m_cycletime = .001;
 bool oldstate = false;
 bool newstate = false;

 mcDeviceRegister(m_cid, m_id, "Sim0", "Simulation Device", DEV_TYPE_MOTION | DE
```

```
 mcIoRegister(m_hDev, "Input0", "Input0", IO_TYPE_INPUT, &m;_Input0);
 mcIoGetType(m_Input0, &type;);
// type will equal IO_TYPE_INPUT
}
```

mcIoGetUserData

C/C++ Syntax:

```
mcIoGetUserData(
 HMCIO hIo,
 void **data);
```

LUA Syntax:

```
N/A
```

Description: Retrieve the type user data pointer associated with the I/O object..

Parameters:

Parameter	Description
hIo	The IO handle.
data	The address of a pointer to receive the user data.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The **hIo** parameter was 0 or invalid.

Notes:
The hIo parameter **MUST** be a valid IO handle. Otherwise, the function will crash.

Usage:

```
struct myData {
 int myPortNumber;
 int myPinNumber;
};

void GetUserData(void)
{
 MINSTANCE mInst = 0;
 HMCIO hIo;
 void *data;
 if (mcIoGetHandle(mInst, "Sim0/Input0", &hIo;) == MERROR_NOERROR) {
  mcIoGetUserData(hIo, &data;);
// type cast the pointer
```

```
  struct myData *mData = (struct myData *)data;
 }
}
```

mcIoIsEnabled

C/C++ Syntax:

```
mcIoIsEnabled(
 HMCIO hIo,
 BOOL *enabled);
```

LUA Syntax:

```
enabled, rc = mcIoIsEnabled(
 number hIo)
```

Description: Check to see if the IO object has been enabled.

Parameters:

Parameter	Description
hSig	A sginal handle obtained from mcIoGetHandle().
state	The address of a BOOL to receive the enabled state.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_IO_NOT_FOUND	The IO object was not found.

Notes:
The hIo parameter **MUST** be a valid IO handle. Otherwise, the function will return MERROR_IO_NOT_FOUND.

Usage:

```
HMCIO hIo;
char *path = "Sim0/Input0";
BOOL enabled;
mcIoGetHandle(m_cid, path, &hIo;);//Get the handle
mcIoIsEnabled(hIo , &enabled;);//Get the enabled state of the IO
```

mcIoRegister

C/C++ Syntax:

```
mcIoRegister(
 HMCDEV hDev,
 const char *IoName,
 const char *IoDesc,
 int Type,
 HMCIO *hIo);
```

LUA Syntax:

```
N/A
```

Description: Register the I/O object in the Mach core.

Parameters:

Parameter	Description
hDev	Pointer to the device.
IoName	Name of the IO point.
IoDesc	Description of the IO.
Type	IO_TYPE_NONE, IO_TYPE_INPUT, IO_TYPE_OUTPUT, IO_TYPE_ANA_INPUT, IO_TYPE_ANA_OUTPUT.
hIo	Address of the IO point

Returns:

Return Code	Description
MERROR_NOERROR	No Error.

Notes:
The hDev parameter **MUST** be a valid device handle. Otherwise, the function will crash.

Usage:

```
simControl::simControl(MINSTANCE mInst, HMCPLUG id)
{
 m_cid = mInst;
 m_id = id;
 m_timer = new simTimer(this);
```

```
 m_cycletime = .001;

 mcDeviceRegister(m_cid, m_id, "Sim0", "Simulation Device", DEV_TYPE_MOTION | DEV

 mcIoRegister(m_hDev, "Input0", "Input0", IO_TYPE_INPUT, &m;_Input0);
 mcIoRegister(m_hDev, "Output0", "Output0", IO_TYPE_OUTPUT, &m;_Output0);

 if (!m_timer->Start(10, wxTIMER_ONE_SHOT)) {
  wxMessageBox(wxT("Timer could not start!"), wxT("Timer"));
 }
}
```

mcIoSetDesc

C/C++ Syntax:

```
mcIoSetDesc(
 HMCIO hIo,
 const char *desc);
```

LUA Syntax:

```
rc = mc.mcIoSetDesc(
 number hIo,
 string desc)
```

Description: Sets the decription of a registerd I/O object.

Parameters:

Parameter	Description
hIo	The I/O handle.
desc	The description to apply to the I/O object.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The hIo parameter was 0 or invalid.

Notes:
The hIo parameter **MUST** be a valid IO object handle. Otherwise, the function will crash.

Usage:

```
HMCIO hIo;
char *path = "Sim0/Input0";

if (mcIoGetHandle(m_cid, path, &hIo;) == MERROR_NOERROR) {
 mcIoSetDesc(hIo, "my mew description);
}
```

mcIoSetName

C/C++ Syntax:

```
mcIoSetName(
 HMCIO hIo,
 const char *name);
```

LUA Syntax:

```
rc = mc.mcIoSetName(
 number hIo,
 string name)
```

Description: Sets the name of a registerd I/O object.

Parameters:

Parameter	Description
hIo	The I/O handle.
name	The name to apply to the I/O object.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The hIo parameter was 0 or invalid.

Notes:
The hIo parameter **MUST** be a valid IO object handle. Otherwise, the function will crash.

Usage:

```
HMCIO hIo;
char *path = "Sim0/Input0";

if (mcIoGetHandle(m_cid, path, &hIo;) == MERROR_NOERROR) {
 mcIoSetName(hIo, "Limit0++");
// The name is changed from Input0 to Limit0++
}
```

mcIoSetState

C/C++ Syntax:

```
mcIoSetState(
 HMCIO hIo,
 bool state);
```

Description: Sets the state of a registerd I/O object.

Parameters:

Parameter	Description
hIo	The I/O object handle.
state	The state of I/O object

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The hIo parameter was 0.

Notes:
The hIo parameter **MUST** be a valid IO object handle. Otherwise, the function will crash.
A value of true means the I/O object is in a high state. This function should only be used upon a state change. It is a programming error otherwise and will cause messages to SPAM the core and GUI.

Usage:

```
simControl::simControl(MINSTANCE mInst, HMCPLUG id)
{
 m_cid = mInst;
 m_id = id;
 m_timer = new simTimer(this);
 m_cycletime = .001;
 bool oldstate = false;
 bool newstate = false;

 mcDeviceRegister(m_cid, m_id, "Sim0", "Simulation Device", DEV_TYPE_MOTION | DE

 mcIoRegister(m_hDev, "Input0", "Input0", IO_TYPE_INPUT, &m;_Input0);
 mcIoGetState(m_Input0, &oldstate;);
 newstate = simGetIO(0);
 if (newstate != oldstate) {
```

```
  mcIoSetState(m_Input0, newstate);
 }
}
```

mcIoSetType

C/C++ Syntax:

```
mcIoSetType(
 HMCIO hIo,
 int type);
```

LUA Syntax:

```
rc = mcIoSetType(
 number hIo
 number type)
```

Description: Retrieve the type (input or output) the IO object.

Parameters:

Parameter	Description
hIo	The I/O object handle.
type	An integer specifying the type of the I/O object. (IO_TYPE_INPUT or IO_TYPE_OUTPUT)

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The **hIo** parameter was 0 or invalid.

Notes:
The hIo parameter **MUST** be a valid IO handle. Otherwise, the function will crash.

Usage:

```
simControl::simControl(MINSTANCE mInst, HMCPLUG id)
{
 m_cid = mInst;
 m_id = id;
 m_timer = new simTimer(this);
 m_cycletime = .001;
 bool oldstate = false;
 bool newstate = false;

 mcDeviceRegister(m_cid, m_id, "Sim0", "Simulation Device", DEV_TYPE_MOTION | DE
```

```
 mcIoRegister(m_hDev, "Input0", "Input0", IO_TYPE_INPUT, &m;_Input0);
 mcIoSetType(m_Input0, IO_TYPE_OUTPUT);
// change the type to IO_TYPE_OUTPUT
}
```

mcIoSetUserData

C/C++ Syntax:

```
mcIoSetUserData(
 HMCIO hIo,
 void **data);
```

LUA Syntax:

```
N/A
```

Description: Retrieve the type user data pointer associated with the I/O object..

Parameters:

Parameter	Description
hIo	The IO handle.
data	The address of a pointer to receive the user data.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The **hIo** parameter was 0 or invalid.

Notes:
The hIo parameter **MUST** be a valid IO handle. Otherwise, the function will crash.

Usage:

```
struct myData {
 int myPortNumber;
 int myPinNumber;
};

struct myData data;

void GetUserData(void)
{
 MINSTANCE mInst = 0;
 HMCIO hIo;
 struct myData data;
 data.myPortNumber = 1;
```

```
 data.myPinNumber = 12;
 if (mcIoGetHandle(mInst, "Sim0/Input0", &hIo;) == MERROR_NOERROR) {
  mcIoSetUserData(hIo, &data;);
 }
}
```

mcIoSyncSignal

C/C++ Syntax:

```
mcIoSyncSignal(
 HMCIO hIo,
 BOOL state);
```

LUA Syntax:

```
N/A
```

Description: A special function that I/O plugins can use to force the synchronization of a Mach output signal.

Parameters:

Parameter	Description
hIo	The I/O object handle.
state	An BOOL specifying the state to sync to the I/O objects mapped core output signal

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The **hIo** parameter was 0 or invalid.

Notes:
The hIo parameter **MUST** be a valid IO handle. Otherwise, the function will crash. This function is primarily used to synchronize the core output signal when setting/clearing outputs are coordinated with machine motion. (M63/M63)

Usage:

```
N/A
```

mcIoUnregister

C/C++ Syntax:

```
mcIoUnregister(
 HMCDEV hDev,
 HMCIO hIo);
```

LUA Syntax:

```
N/A
```

Description: Unregister the I/O object in the Mach core.

Parameters:

Parameter	Description
hDev	Pointer to the device.
hIo	The /O object handle to remove from the core..

Returns:

Return Code	Description
MERROR_NOERROR	No Error.

Notes:
The hDev parameter **MUST** be a valid device handle. Otherwise, the function will crash.

Usage:

N/A

mcJogGetAccel

C/C++ Syntax:

```
int mcJogGetAccel(
  MINSTANCE mInst,
  int axisId,
  double *percent);
```

LUA Syntax:

```
percent, rc = mc.mcJogGetAccel(
  number mInst,
  number axis)
```

Description: Get the current jog acceleration value for the give aixs as a percentage of max velocity.

Parameters: (percent, The address of a double to receive the current jog acceleration percentage.

Parameter	Description
mInst	The controller instance.
axisId	An integer specifying the axis.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.
MERROR_INVALID_ARG	**percent** is NULL.

Notes:

Usage:

```
// Get the jog accel percentage for the X axis.
MINSTANCE mInst = 0;
double accel;
int rc = mcJogGetAccel(mInst, X_AXIS, &accel;);
```

mcJogGetFollowMode

C/C++ Syntax:

```
mcJogGetFollowMode(
 MINSTANCE mInst,
 double *mode_on);
```

Description: Get the jog follow state of the toolpath. This is when the tool path will auto pan to follow the tool on the screen.

Parameters:

Parameter	Description
mInst	The controller instance.
mode_on	A pointer to an int to receive the jog follow state (MC_ON, MC_OFF).

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
No Error.	
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
The mInst parameter was out of range.	

Notes:
This call is used to help display the state to the user.

Usage:

```
int mInst=0;
double jfstat=0;
mcJogGetFollowMode(mInst, &jfstate;);
```

mcJogGetInc

C/C++ Syntax:

```
int mcJogGetInc(
  MINSTANCE mInst,
  int axisId,
  double *increment);
```

LUA Syntax:

```
increment, rc = mc.mcJogGetInc(
  number mInst,
  number axisId)
```

Description: Returns the current jog increment for the given axis.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.
MERROR_INVALID_ARG	**increment** is NULL.

Notes:

Usage:

```
// Get the jog increment for the X axis.
MINSTANCE mInst = 0;
double inc;
int rc = mcJogGetIncAccel(mInst, X_AXIS, &inc;);
```

mcJogGetRate

C/C++ Syntax:

```
int mcJogGetRate(
  MINSTANCE mInst,
  int axisId,
  double *percent);
```

LUA Syntax:

```
percent, rc = mc.mcJogGetRate(
  number mInst,
  number axisId)
```

Description: Returns the current jog rate as a percentage of max velocity.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	An integer specifying the axis.
percent	The address of a double to receive the jog rate percentage.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.
MERROR_INVALID_ARG	**percentage** is NULL.

Notes:

Usage:

```
// Get the current jog rate for the X axis
MINSTANCE mInst = 0;
double jogRate;
int rc = mcJogGetRate(mInst, X_AXIS, &jogRate;);
```

mcJogGetVelocity

C/C++ Syntax:

```
int mcJogGetVelocity(
  MINSTANCE mInst,
  int axisId,
  double *vel);
```

LUA Syntax:

```
vel, rc = mc.mcJogGetVelocity(
  number mInst,
  number axisId),
```

Description: Get the current jog velocity setting in units per minute.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis from which to get the jog velocity.
vel	The address of a double to receive the current jog velocity.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Changing the jog rate (percentage of max velocity) will affect the return value.

Usage:

```
// Get the current jog velocity setting for the X axis
MINSTANCE mInst = 0;
double jogVel;
int rc = mcJogGetVelocity(mInst, X_AXIS, &jogVel;);
```

mcJogIncStart

C/C++ Syntax:

```
int mcJogIncStart(
  MINSTANCE mInst,
  int axisId,
  double dist);
```

LUA Syntax:

```
rc = mc.mcJogIncStart(
  number mInst,
  number axisId,
  number dist)
```

Description: Start an Incermetnal jog move.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis to start jogging.
dist	The incremental distance to jog the axis (added to current goal position).

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Inc jog or position any axis. The axis will use the MaxVel and Accel settings to get into position.

Usage:

```
MINSTANCE mInst = 0;
int axis = Z_AXIS;
double Joginc = .100;
// Jog controller 0 .1 in the Z axis.
```

```
mcJogIncStart(mInst, axis, Joginc);
```

mcJogIncStop

C/C++ Syntax:

```
int mcJogIncStop(
  MINSTANCE mInst,
  int axisId,
  double incr);
```

LUA Syntax:

```
rc = mc.mcJogIncStop(
  number mInst,
  number axisId,
  number incr)
```

Description: Stop an incremental jog at the closest increment.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis to stop jogging.
inc	The increment on which to to stop the axis.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
This function will stop an axis on an increment. This feature is like a digital detent to allow an axis to stop at some set increment. The original use is to stop an axis that is over loaded with inc jog requests.

Usage:

```
// Stop the incremental jog on the Z axis.
MINSTANCE mInst = 0;
```

```
int axis = Z_AXIS;
double Joginc = .100;
int rc = mcJogIncStop(mInst, axis, Joginc);
```

mcJogIsJogging

C/C++ Syntax:

```
int mcJogIsJogging(
  MINSTANCE mInst,
  int axisId,
  BOOL *jogging);
```

LUA Syntax:

```
jogging, rc = mc.mcJogIsJogging(
  number mInst,
  number axisId)
```

Description: Determine if the given axis is jogging.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	An integer specifying the axis.
jogging	The address of a BOOL to receive the axis jog state.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
None.

Usage:

```
// See if the X axis is jogging.
MINSTANCE mInst = 0;
BOOL jogging = FALSE;
int rc = mcJogIsJogging(mInst, X_AXIS, &jogging;);
```

mcJogSetAccel

C/C++ Syntax:

```
int mcJogSetAccel(
  MINSTANCE mInst,
  int axisId,
  double percent);
```

LUA Syntax:

```
rc = mc.mcJogSetAccel(
  number mInst,
  number axisId,
  double percent)
```

Description: Set the jog accel as a percentage of the axis' maximum velocity.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	An integer spcifying the axis.
percent	A double specifying the acceleration percentage.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:

Usage:

```
// Set the X axis jog accel percentage to 75%.
MINSTANCE mInst = 0;
int mcJogSetAccel(mInst, X_AXIS, 75);
```

mcJogSetFollowMode

C/C++ Syntax:

```
mcJogSetFollowMode(
 MINSTANCE mInst,
 double mode_on);
```

Description: Set the jog follow mode of the toolpath.

Parameters:

Parameter	Description
mInst	The controller instance.
mode_on	Sets Jog follow mode to be on or off with (MC_ON, MC_OFF).

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
No Error.	
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
The mInst parameter was out of range.	

Notes:
This can be set at anytime.

Usage:

```
int mInst=0;
mcJogGetFollowMode(mInst, MC_ON);
```

mcJogSetInc

C/C++ Syntax:

```
int mcJogSetInc(
  MINSTANCE mInst,
  int axisId,
  double increment);
```

LUA Syntax:

```
rc = mc.mcJogSetInc(
  number mInst,
  number axisId,
  number increment)
```

Description: Set the current jog increment for the give axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	An integer specifying the axis.
increment	A double specifying the desired increment.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
None.

Usage:

```
// Set the X axis jog increment to .010"
MINSTANCE mInst = 0;
int rc = mcJogSetInc(mInst, X_AXIS, .010);
```

mcJogSetRate

C/C++ Syntax:

```
int mcJogSetRate(
  MINSTANCE mInst,
  int axisId,
  double percent);
```

LUA Syntax:

```
rc = mc.mcJogSetRate(
  number mInst,
  number axisId,
  number percent)
```

Description: Set the jog rate of the given axis as a percentage of the axis' maximum velocity.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis to stop jogging.
percent	The jog rate percentage.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
None

Usage:

```
// Set the jog rate to 75% of the Z axis maximum velocity.
MINSTANCE mInst = 0;
int axis = Z_AXIS;
int rc = mcJogSetRate(mInst, axis, 75);
```

mcJogSetTraceEnable

C/C++ Syntax:

```
int mcJogSetTraceEnable(
  MINSTANCE mInst,
  BOOL enable);
```

LUA Syntax:

```
rc = mc.mcJogSetTraceEnable(
  number mInst,
  number enable)
```

Description: Enable or disable tool path jog tracing.

Parameters:

Parameter	Description
mInst	The controller instance.
enable	A BOOL secifying tool path jog tracing.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None

Usage:

```
//Set jog tracing.
MINSTANCE mInst = 0;
int rc = mcJogSetTraceEnable(mInst, TRUE);
```

mcJogSetType

C/C++ Syntax:

```
int mcJogSetType(MINSTANCE mInst, int axisId, int type);
```

LUA Syntax:

```
rc = mc.mcJogSetType(
  number mInst,
  number axisId,
  number type)
```

Description: Set the jog operation type for the given axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	An integer specifying the axis.
type	An integer specifying the jog type. (MC_JOG_TYPE_VEL or MC_JOG_TYPE_INC)

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
None

Usage:

```
// Set the X axis jog type to velocity mode.
MINSTANCE mInst = 0;
int rc = mcJogSetType(mInst, X_AXIS, MC_JOG_TYPE_VEL);
```

mcJogVelocityStart

C/C++ Syntax:

```
int mcJogVelocityStart(
  MINSTANCE mInst,
  int axisId,
  double dir);
```

LUA Syntax:

```
rc = mc.mcJogVelocityStart(
  number mInst,
  number axisId,
  number dir);
```

Description: Start a velocity jog on the given axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axis	The axis to jog.
dir	An integer specifying the direction of the jog. (MC_JOG_POS, MC_JOG_NEG, MC_JOG_STOP)

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.
MERROR_NOT_NOW	The operation could not be completed at this time.

Notes:
Specifying MC_JOG_STOP for **dir** is the same as calling mcJogVelocityStop().

Usage:

```
int mInst=0;
```

```
int axis = Z_AXIS;
int rc = mcJogVelocityStart(mInst, axis, MC_JOG_POS); // Start Z axis jogging fo
if (rc == MERROR_NOERROR) {
 Sleep(5000);
}
rc = mcJogVelocityStart(mInst, axis, MC_JOG_POS); // Reverse axis and jog the ax
if (rc == MERROR_NOERROR) {
 Sleep(5000);
}
mcJogVelocityStop(mInst, axis); // Stop the axis.
```

mcJogVelocityStop

C/C++ Syntax:

```
mcJogVelocityStop(
  MINSTANCE mInst,
  int axisId);
```

LUA Syntax:

```
mcJogVelocityStop(
  number mInst,
  number axisId)
```

Description: Stop a velocity jog on an the given axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axisId	The axis to stop.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Stops the axis with respect to acceleration settings of the slowest motor that is part of the axis.

Usage:

```
MINSTANCE mInst = 0;
int axis = Z_AXIS;
int rc;
// Jog axis at 10 % max velocity.
rc = mcJogSetRate(mInst, axis, 10);
rc = mcJogVelocityStart(mInst, axis, MC_JOG_POS);
if (rc == MERROR_NOERROR) {
 Sleep(5000);
}
```

```
// Stop the axis.
rc = mcJogVelocityStop(mInst, axis);
```

mcMotionClearPlanner

C/C++ Syntax:

```
int mcMotionClearPlanner(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcMotionClearPlanner(
  number mInst)
```

Description: Clears the motion planner of all previously planned moves.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
This function is useful when probing and the probe has struck before the planned moves are exhausted.

Usage:

```
// Clear the planner.
MINSTANCE mInst = 0;
int rc = mcMotionClearPlanner(mInst);
```

mcMotionCyclePlanner

C/C++ Syntax:

```
int mcMotionCyclePlanner(MINSTANCE mInst);
```

LUA Syntax:

```
N/A
```

Description: Cycle the core planner.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
This function is depricated in favor of mcMotionCyclePlannerEx().

Usage:

```
None.
```

mcMotionCyclePlannerEx

C/C++ Syntax:

```
int mcMotionCyclePlannerEx(
  MINSTANCE mInst,
  execution_t *exInfo);
```

LUA Syntax:

```
N/A
```

Description: Cycles the core planner to produce one time slice of movement information.

Parameters:

Parameter	Description
mInst	The controller instance.
exInfo	The address of an execution_t struct that receives the movement information.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**exInfo** is NULL.

Notes:
This is the main function a motion plugin uses to drive motion.

Usage:

```
// Cycle the core planner.
MINSTANCE mInst = 0;
execution_t exInfo;
int rc = mcMotionCyclePlannerEx(mInst, &exInfo;);
// For more information, see the Sim plugin example.
```

mcMotionGetAbsPos

C/C++ Syntax:

```
int mcMotionGetAbsPos(
  MINSTANCE mInst,
  int motorId,
  double *val);
```

LUA Syntax:

```
N/A
```

Description: Retrieves the last planned absolute motor position for the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	An integer specifying the motor.
val	The address of a double to receive the position.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
This function has been depricated in favor of using mcMotionCyclePlannerEx().

Usage:

```
None.
```

mcMotionGetAbsPosFract

C/C++ Syntax:

```
int mcMotionGetAbsPosFract(
  MINSTANCE mInst,
  int motorId,
  double *val);
```

LUA Syntax:

```
N/A
```

Description: Retrieves the last planned absolute motor position for the given motor (with fractions).

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	An integer specifying the motor.
val	The address of a double to receive the position.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
This function has been depricated in favor of using mcMotionCyclePlannerEx().

Usage:

```
None.
```

mcMotionGetBacklashAbs

C/C++ Syntax:

```
int mcMotionGetBacklashAbs(
  MINSTANCE mInst,
  int motorId,
  double *pos);
```

LUA Syntax:

```
N/A
```

Description: Retrieves the last planned backlast amount for the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	An integer specifying the motor.
pos	The address of a double to receive the position.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
This function has been depricated in favor of using mcMotionCyclePlannerEx().

Usage:

```
None.
```

mcMotionGetBacklashInc

C/C++ Syntax:

```
int mcMotionGetBacklashInc(
  MINSTANCE mInst,
  int motorId,
  double *pos);
```

LUA Syntax:

```
N/A
```

Description: Retrieves the last planned backlast amount for the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	An integer specifying the motor.
pos	The address of a double to receive the position.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
This function has been depricated in favor of using mcMotionCyclePlannerEx().

Usage:

```
None.
```

mcMotionGetIncPos

C/C++ Syntax:

```
int mcMotionGetIncPos(
  MINSTANCE mInst,
  int motorId,
  double *val);
```

LUA Syntax:

```
N/A
```

Description: Retrieves the last planned incremental motor position for the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	An integer specifying the motor.
val	The address of a double to receive the position.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
This function has been depricated in favor of using mcMotionCyclePlannerEx().

Usage:

```
None.
```

mcMotionGetMoveID

C/C++ Syntax:

```
int mcMotionGetMoveID(
  MINSTANCE mInst,
  int motorId,
  long *val);
```

LUA Syntax:

```
N/A
```

Description: Retrieves the last planned move ID for the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	An integer specifying the motor.
val	The address of a long to receive the value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
This function has been depricated in favor of using mcMotionCyclePlannerEx().

Usage:

```
None.
```

mcMotionGetPos

C/C++ Syntax:

```
int mcMotionGetPos(
  MINSTANCE mInst,
  int motorId,
  double *pos);
```

LUA Syntax:

```
pos, rc = mc.mcMotionGetPos(
  number mInst,
  number motorId)
```

Description: Retrieves the current motor pos in counts.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	An integer specifying the motor.
pos	The address of a double to receive the position.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
None.

Usage:

```
// Get the motor 0 position.
MINSTANCE mInst = 0;
double pos;
int rc = mcMotionGetPos(mInst, 0, &pos;);
```

mcMotionGetProbeParams

C/C++ Syntax:

```
int mcMotionGetProbeParams(
  MINSTANCE mInst,
  probe_t *probeInfo);
```

LUA Syntax:

```
N/A
```

Description: Get the probing parameters from the core.

Parameters:

Parameter	Description
mInst	The controller instance.
probeInfo	The address of a prob_t struct to receive the probing parameters.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
A motion device registered with DEV_TYPE_PROBE2 must be defined in order to use this function.

Usage:

```
// Get the probing parameters from the core.
MINSTANCE mInst = 0;
probe_t pInfo;
int mcMotionGetProbeParams(mInst, &pInfo;);
```

mcMotionGetRigidTapParams

C/C++ Syntax:

```
int mcMotionGetRigidTapParams(
  MINSTANCE mInst,
  tap_t *tapInfo);
```

LUA Syntax:

```
N/A
```

Description: Retrieve the tapping parameters from the core.

Parameters:

Parameter	Description
mInst	The controller instance.
tapInfo	The address of a tap_t struct to receive the tapping parameters.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
A motion device registered with DEV_TYPE_TAP2 must be defined in order to use this function.

Usage:

```
// Get the tpping parameters from the core.
MINSTANCE mInst = 0;
tap_t tapInfo;
int rc = mcMotionGetRigidTapParams(mInst, &tapInfo;);
```

mcMotionGetSyncOutput

C/C++ Syntax:

```
int mcMotionGetSyncOutput(
  MINSTANCE mInst,
  int outputQueue,
  HMCIO *hIo,
  BOOL *state);
```

LUA Syntax:

```
N/A
```

Description: Gets a movement coordinated output from the core.

Parameters:

Parameter	Description
mInst	The controller instance.
outputQueue	An integer specifying which output queue.
hIo	The address of a HMCIO variable to receive the I/O handle associated with the output.
state	Te address of a BOOL to receive the desired state of the output.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Test the ex.exOutputQueue flag to see if there are any coordinated output to be set/cleared. The **outputQueue** parameter comes from ex.ex.exOutputQueue.

Usage:

```
// Retrieve the coordinated outputs/
MINSTANCE mInst = 0;
execution_t ex;
mcMotionCyclePlannerEx(mInst, &ex;);
```

```
if (ex.exOutputQueue != EX_NONE) {
 HMCIO hIo;
 BOOL state;
 while (mcMotionGetSyncOutput(m_cid, ex.exOutputQueue, &hIo;, &state;) == MERROR_
  // Pair setting the output along with the moves in (execution_t)ex.
 }
}
```

mcMotionGetThreadParams

C/C++ Syntax:

```
int mcMotionGetThreadParams(
  MINSTANCE mInst,
  thread_t *threadInfo);
```

LUA Syntax:

Description:

Parameters:

Parameter	Description
mInst	The controller instance.
threadInfo	The address of a thread_t struct to receive the threading parameters.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_IMPLEMENTED	The motion device does not support DEV_TYPE_THREAD2 type threading.

Notes:
A motion device registered with DEV_TYPE_THREAD2 is required.

Usage:

```
//
MINSTANCE mInst = 0;
thread_t threadInfo;
int mcMotionGetThreadParams(mInst, &threadInfo;);
```

mcMotionGetThreadingRate

C/C++ Syntax:

```
int mcMotionGetThreadingRate(
  MINSTANCE mInst,
  double *ratio);
```

LUA Syntax:

```
N/A
```

Description: Get the current threading ration from the core.

Parameters:

Parameter	Description
mInst	The controller instance.
ratio	The address to a double to receive the threading ratio.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

Usage:

```
// Get the current threading rate from the core.
MINSTANCE mInst = 0;
double ratio;
int rc = mcMotionGetThreadingRate(mInst, :);
```

mcMotionGetVel

C/C++ Syntax:

```
int mcMotionGetVel(MINSTANCE mInst, int motorId, double *velocity);
```

LUA Syntax:

```
velocity, rc = mc.mcMotionGetVel(
  number mInst,
  number motorId)
```

Description: Retrieve the current motor velocity in counts per second squared.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	An integer specifying the motor ID.
velocity. The address of a double to receive the motor velocity.	

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.
MERROR_INVALID_ARG	**velocity** is NULL or **motorId** is less than 0.

Notes:
None.

Usage:

```
// Get the motor velocity for motor 1
MINSTANCE mInst = 0;
double vel;
int rc = mcMotionGetVel(mInst, 1, &vel;);
```

mcMotionSetCycleTime

C/C++ Syntax:

```
int mcMotionSetCycleTime(
  MINSTANCE mInst,
  double secs);
```

LUA Syntax:

```
N/A
```

Description: Set the time slice of a cycle.

Parameters:

Parameter	Description
mInst	The controller instance.
secs	A double specifying the cycle time slice.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Decimal values are accepted. e.g. .001 is equal to 1 millisecond.

Usage:

```
// Set the cycle time slice.
MINSTANCE mInst = 0;
int mcMotionSetCycleTime(mInst, .001);
```

mcMotionSetMoveID

C/C++ Syntax:

```
int mcMotionSetMoveID(
  MINSTANCE mInst,
  int id);
```

LUA Syntax:

```
N/A
```

Description: Set the time slice of a cycle.

Parameters:

Parameter	Description
mInst	The controller instance.
id	Set the currently executing movement ID.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Ideally, the motion control device should report the currently executing movement.

Usage:

```
// Set the currently executing movement ID.
MINSTANCE mInst = 0;
int moveId = 10001; // Should be obtained from the motion controller.
int mcMotionSetMoveID(mInst, moveId);
```

mcMotionSetPos

C/C++ Syntax:

```
int mcMotionSetPos(
  MINSTANCE mInst,
  int motorId
  double val);
```

LUA Syntax:

```
N/A
```

Description: Report the position in counts to the core for the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
MotorId	An integer specifying the motor.
val	A double specifying the motor position in counts.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
The motion control device should report the motor positions to the core so that the core can synchronize events based on the current position.

Usage:

```
// Set the motor position for motor 1.
MINSTANCE mInst = 0;
motorCounts = 324255; // Should be retrieved from the motion controller.
int mcMotionSetPos(mInst, 1, motorCounts);
```

mcMotionSetProbeComplete

C/C++ Syntax:

```
int mcMotionSetProbeComplete(
  MINSTANCE mInst);
```

LUA Syntax:

```
N/A
```

Description: Inform the core that a probe operation is complete.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Calling this function assumes that a probe stike has happened and that the probed positions have been reproted.

Usage:

```
// Complete a probe operation.
MINSTANCE mInst = 0;
int rc = mcMotionSetProbeComplete(mInst);
```

mcMotionSetProbePos

C/C++ Syntax:

```
int mcMotionSetProbePos(
  MINSTANCE mInst,
  int motorId,
  double val);
```

LUA Syntax:

```
N/A
```

Description: Report the probed position for the given motor to the core.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	An integer specifying the motor.
val	A double specifying the probed motor position in counts.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
When the probe strikes, the position needs to be reported to the core.

Usage:

```
// Set the probed position for motor 0.
MINSTANCE mInst = 0;
double probedPos = 2314134; // Should be reteived from motion controller latch re
int rc = mcMotionSetProbePos(mInst, 0, double val);
```

mcMotionSetStill

C/C++ Syntax:

```
int mcMotionSetStill(
  MINSTANCE mInst,
  int motorId);
```

LUA Syntax:

```
N/A
```

Description: Report when the given motor is still.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	An integer specifying the motor.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
The core will request a stop report in order to synchronize events with the end of movement. This is a very important thing to get correct in a motion plugin. This is how the core will know to stop waiting on M code or any other G code that breaks the CV chain.

Usage:

```
// Motor stop report example.
MINSTANCE mInst = 0;
execution_t ex;
mcMotionCyclePlannerEx(m_cid, &ex;);
switch(ex.exType) {
case EX_STOP_REQ:
 // See if we need to report when the motors are still.
```

```
 for (i = 0; i < 8; i++) {
  if (ex.exMotors[i].reportStopped == TRUE) {
   // Report when this motor has completed all previous moves!
  }
 }
 break;
 ...
}
```

mcMotionSetThreadingRate

C/C++ Syntax:

```
int mcMotionSetThreadingRate(
  MINSTANCE mInst,
  double ratio);
```

LUA Syntax:

```
N/A
```

Description: Set the threading skew ratio.

Parameters:

Parameter	Description
mInst	The controller instance.
ratio	A double specifying the skew ratio.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Set the threading ratio.
MINSTANCE mInst = 0;
int rc = mcMotionSetThreadingRate(mInst, 1.2);
```

mcMotionSetVel

C/C++ Syntax:

```
int mcMotionSetVel(
  MINSTANCE mInst,
  int motorId,
  double velocity);
```

LUA Syntax:

```
N/A
```

Description: Report the velocity (in counts per second squared) to the core for the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	An integer specifying the motor.
velocity	A double specifying the velocity in counts per second squared.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
Motion plugins use this function to report motor velocities to the core.

Usage:

```
// Report the velocity for motor 0.
MINSTANCE mInst = 0;
double vel = 2342420;// Should be obtained from the motion controller.
int mcMotionSetVel(mInst, 0, vel);
```

mcMotionSync

C/C++ Syntax:

```
int mcMotionSync(
  MINSTANCE mInst);
```

LUA Syntax:

```
N/A
```

Description: Synchronize the core planners with the last reported motor positions.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
If movement has been done outside of direction from the core, the planners must be synced to the new motor positons.

Usage:

```
// Synch core planner positions with the last reported motor positions.
MINSTANCE mInst = 0;
int rc = mcMotionSync(mInst);
```

mcMotionThreadComplete

C/C++ Syntax:

```
int mcMotionThreadComplete(
  MINSTANCE mInst);
```

LUA Syntax:

```
N/A
```

Description: Inform the core that the threading operation is complete.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
If the motion controller is profiling all of the threading moves, the core must be informed when the threading operation is complete.

Usage:

```
// Report that the threading op is complete.
MINSTANCE mInst = 0;
int rc = mcMotionThreadComplete(mInst);
```

mcMotorGetAxis

C/C++ Syntax:

```
int mcMotorGetAxis(
  MINSTANCE mInst,
  int motorId,
  int *axisId)
```

LUA Syntax:

```
axisId, rc = mc.mcMotorGetAxis(
  number mInst,
  number motorId)
```

Description:

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	An integer specifying the motor.
axisId	The address of an interger to receive the motor's mapped axis.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.
MERROR_AXIS_NOT_FOUND	No axis has the specified motor mapped.

Notes:

axisId will be -1 if the function returns MERROR_AXIS_NOT_FOUND.

Usage:

```
// Get the parent axis for motor 0.
MINSTANCE mInst = 0;
int axisId = -1;
int rc = mcMotorGetAxis(mInst, 0, &axisId;);
```

mcMotorGetInfoStruct

C/C++ Syntax:

```
mcMotorGetInfoStruct(
  MINSTANCE mInst,
  int motorId,
  motorinfo_t *minf);
```

LUA Syntax:

```
N/A
```

Description: Retrieve the motor information of the motor in one call.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	The motor id.
minf	The address of a motorinfo_t struct to receive the motor settings.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.
MERROR_INVALID_ARG	**minf** cannot be NULL.

Notes:
Retrieve all the motor settings in one call.
To change any of the settings mcMotorSetInfoStruct() may be called.

```
struct motorinfo {
 double CountsPerUnit;     // Number of encoder counts or steps per unit.
 double MaxVelocity;       // Max velocity of the axis.
 double MaxAcceleration;   // Max rate to accelerate.
 BOOL   Reverse;           // Is the axis reversed?
```

```
 double BacklashAmount;   // The amount of backlash in counts.
 double CurrentVelocity;  // The speed the axis is moving, This could be reporte
 int    CurrentPosition;  // The Current Position (From the motion device).
 BOOL   Homed;            // True if the axis has been homed.
 long   SoftMaxLimit;     // Count for the max travel.
 long   SoftMinLimit;     // Count for the min travel.
 BOOL   CanHome;          // Can this motor home?
 BOOL   Enabled;          // Is this motor enabled?
 long   EnableDelay;      // ms to delay the enable signal for this motor.
 int    AxisId;           // -1 if no axis has mapped this motor.
};
typedef struct motorinfo motorinfo_t;
```

Usage:

```
MINSTANCE mInst = 0;
int m = 2;
motorinfo_t minf;
mcMotorGetInfoStruct(mInst, m, &minf;); // Get data for motor 2.
```

mcMotorGetMaxAccel

C/C++ Syntax:

```
int mcMotorGetMaxAccel(
  MINSTANCE mInst,
  int motorId,
  double *maxAccel);
```

LUA Syntax:

```
maxAccel, rc = mc.mcMotorGetMaxAccel(
  number mInst,
  number motorId)
```

Description: Retrieve the maximum acceleration value for the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	An integer specifying the motor.
maxAccel	The address of a double to receive the maximum acceleration value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
None.

Usage:

```
// Get the max accel for motor 0.
MINSTANCE mInst = 0;
double maxAccel = 0.0;
```

```
int rc = mcMotorGetMaxAccel(mInst, 0, &maxAccel;);
```

mcMotorGetMaxVel

C/C++ Syntax:

```
int mcMotorGetMaxVel(
  MINSTANCE mInst,
  int motorId,
  double *maxVel)
```

LUA Syntax:

```
maxVel, rc = mc.mcMotorGetMaxVel(
  number mInst,
  number motorId)
```

Description: Retrieve the maximum velocity for the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	An integer specifying the motor.
maxVel	The address of a double to receive the maximum velocity value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
None.

Usage:

```
// Get the max vel for motor 0.
MINSTANCE mInst = 0;
double maxVel = 0.0;
```

```
int rc = mcMotorGetMaxVel(mInst, 0, &maxVel;);
```

mcMotorGetPos

C/C++ Syntax:

```
int mcMotorGetPos(
  MINSTANCE mInst,
  int motorId,
  double *val);
```

LUA Syntax:

```
val, rc = mc.mcMotorGetPos(
  number mInst,
  number motorId)
```

Description: Retrieve the position of the given motor in counts (fractional counts supported).

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	An integer specifying the motor.
val	The address of a double to receive the motor position.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.
MERROR_INVALID_ARG	**val** cannot be NULL.

Notes:
If the motor is not setup, **val** will contain zero.

Usage:

```
MINSTANCE mInst = 0;
double Motor1Pos = 0;
// Get the position of the Motor 1.
```

```
mcMotorGetPos(mInst, 1, &Motor1Pos;);
```

mcMotorGetVel

C/C++ Syntax:

```
int mcMotorGetVel(
  MINSTANCE mInst,
  int motor,
  double *velocity);
```

LUA Syntax:

```
velocity, rc = mc.mcMotorGetVel(
  number mInst,
  number motor)
```

Description: Retreive the current velocity of a motor in counts per sec.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	An integer specifying the motor.
velocity	The address of a double to receive the motor velocity.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.
MERROR_INVALID_ARG	**velocity** cannot be NULL.

Notes:
Reports the speed of a motor. The returned value can be fractional counts.
If the motor is not found, a value of zero is returned in velocity.

Usage:

```
MINSTANCE mInst = 0;
int m = MOTOR2;
```

```
double CurrentVel = 0;
mcMotorSetVel(mInst, m, &CurrentVel;); // Get the current speed of motor2.
```

mcMotorIsHomed

C/C++ Syntax:

```
int mcMotorIsHomed(
  MINSTANCE mInst,
  int motorId,
  BOOL *homed);
```

LUA Syntax:

```
homed, rc = mc.mcMotorIsHomed(
  number mInst,
  number motorId)
```

Description: Get the homed state of the motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	The motor id.
homed	The address of a BOOL to receive the homed state of a motor. (TRUE/FALSE)

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.
MERROR_INVALID_ARG	**homed** cannot be NULL.

Notes:
Reports the homed state of the motor. The returned value 1 for homed and 0 for not homed.
All child motors linked to axis must be homed for the axis to report as homed.

Usage:

```
MINSTANCE mInst = 0;
int m = MOTOR2;
int Homed = 0;
mcMotorIsHomed(mInst, m, &Homed;); // Get the homed state of motor2.
```

mcMotorIsStill

C/C++ Syntax:

```
int mcMotorIsStill(
  MINSTANCE mInst,
  int motorId,
  BOOL *still);
```

LUA Syntax:

```
still, rc = mc.mcMotorIsStill(
  number mInst,
  number motorId)
```

Description: Set the motor is not moving from motion Motion plugin that owns this motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorID	The motor id.
still	The address of an BOOL to receive the still state.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.
MERROR_INVALID_ARG	**still** cannot be NULL.

Notes:
Motor has its "isStill" when it is not in motion. **still** == TRUE is no movement, **still** == FALSE motor has movment.

Usage:

```
MINSTANCE mInst = 0;
int m = MOTOR0;
```

```
int still = 0;
mcMotorSetStill(mInst, m, &still;); // Get the state if MOTOR0 is still
```

mcMotorMapGetDefinition

C/C++ Syntax:

```
int mcMotorMapGetDefinition(
  MINSTANCE mInst,
  int motorId,
  long *lengthCounts,
  long *numPoints);
```

LUA Syntax:

```
lengthCounts, numPoints, rc = mc.mcMotorMapGetDefinition(
  number mInst,
  number motorId)
```

Description: Retrieve the motor screw map definition.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	The motor id.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.
MERROR_INVALID_ARG	**lengthCounts** and **numPoints** are NULL.

Notes:
None.

Usage:

```
//
MINSTANCE mInst = 0;
```

```
long lengthCounts;
long numPoints;
int rc = mcMotorMapGetDefinition(mInst, 0, &lengthCounts;, &numPoints;);
```

mcMotorMapGetLength

C/C++ Syntax:

```
int mcMotorMapGetLength(
  MINSTANCE mInst,
  int motorId,
  int *length);
```

LUA Syntax:

```
length, rc = mcMotorMapGetLength(
  number mInst,
  number motorId)
```

Description: Retrieve the motor screw map length (length of measured screw in counts).

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	The motor id.
length	the address of an integer to receive the screw length in counts.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.
MERROR_INVALID_ARG	**length** is NULL.

Notes:
None.

Usage:

```
// Get the screw length for motor zero.
MINSTANCE mInst = 0;
```

```
int length;
int rc = mcMotorMapGetLength(mInst, 0, &length;);
```

mcMotorMapGetNPoints

C/C++ Syntax:

```
int mcMotorMapGetNPoints(
  MINSTANCE mInst,
  int motorId,
  int *points);
```

LUA Syntax:

```
points, rc = mc.mcMotorMapGetNPoints(
  number mInst,
  number motorId)
```

Description: Retrieve the number of measurement points in the screw map for the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	The motor id.
points	the address of an integer to receive the number of measurement points.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.
MERROR_INVALID_ARG	**points** is NULL.

Notes:
Depricated. Use mcMotoMapGetPointCount() instead.

mcMotorMapGetPoint

C/C++ Syntax:

```
int mcMotorMapGetPoint(
  MINSTANCE mInst,
  int motorId,
  int point,
  int *error);
```

LUA Syntax:

```
error, rc = mc.mcMotorMapGetPoint(
  numbser mInst,
  numbser motorId,
  numbser point)
```

Description: Retrieves the screw map error for the given motor and point.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	The motor id.
point	An integer specifying the measurement point number.
error	the address of an integer to receive the screw error (in counts).

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.
MERROR_INVALID_ARG	**error** is NULL.

Notes:
None.

Usage:

```
// Get the screw map error for motor 0, measurement point 10.
MINSTANCE mInst = 0;
int screwErr;
int rc = mcMotorMapGetPoint(mInst, 0, 10, &screwErr;);
```

mcMotorMapGetPointCount

C/C++ Syntax:

```
int mcMotorMapGetPointCount(
  MINSTANCE mInst,
  int motorId,
  int *points);
```

LUA Syntax:

```
points, rc = mc.mcMotorMapGetPointCount(
  number mInst,
  number motorId)
```

Description: Retrieve the number of measurement points in the screw map for the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	The motor id.
points	the address of an integer to receive the number of measurement points.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.
MERROR_INVALID_ARG	**points** is NULL.

Notes:
None.

Usage:

```
// Get the number of measurement points.
MINSTANCE mInst = 0;
```

```
int points;
int rc = mcMotorMapGetPointCount(mInst, 0, &points;);
```

mcMotorMapGetStart

C/C++ Syntax:

```
int mcMotorMapGetStart(
  MINSTANCE mInst,
  int motorId,
  int *startPoint);
```

LUA Syntax:

```
startPoint, rc = mc.mcMotorMapGetStart(
  number mInst,
  number motorId)
```

Description: Retrieve the starting point of the motor screw map for the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	The motor id.
startPoint	The address of an integer to receive the screw map starting point.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.
MERROR_INVALID_ARG	**startPoint** is NULL.

Notes:
None.

Usage:

```
// Get the starting point for the motor 0 screw map.
MINSTANCE mInst = 0;
```

```
int startPoint;
int rc = mcMotorMapGetStart(mInst, 0, &startPoint;);
```

mcMotorMapSetDefinition

C/C++ Syntax:

```
int mcMotorMapSetDefinition(
  MINSTANCE mInst,
  int motorId,
  long lengthCounts,
  long numPoints);
```

LUA Syntax:

```
rc = mc.mcMotorMapSetDefinition(
  number mInst,
  number motorId,
  number lengthCounts,
  number numPoints)
```

Description:

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	The motor id.
lengthCounts	An integer specifying the length of the measured screw in counts.
numPoints	An integer specifying the number of measurement points in the screw map.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:

This is the prefered method of defining a motor screw map.

Usage:

```
// Set the screw map definition for motor 0.
MINSTANCE mInst = 0;
int rc = mcMotorMapSetDefinition(mInst, 0, 20000, 20);
```

mcMotorMapSetLength

C/C++ Syntax:

```
int mcMotorMapSetLength(
  MINSTANCE mInst,
  int motorId,
  int length);
```

LUA Syntax:

```
rc = mc.mcMotorMapSetLength(
  number mInst,
  number motorId,
  number length);
```

Description: Set the length of the screw (in counts) for the screw map of the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	The motor id.
length	An integer specifying the length.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.
MERROR_INVALID_ARG	**length** is less <= 0.

Notes:
None.

Usage:

```
// Set the screw length for the motor 0 screw map.
MINSTANCE mInst = 0;
```

```
int rc = mcMotorMapSetLength(mInst, 0, 20000);
```

mcMotorMapSetNPoints

C/C++ Syntax:

```
int mcMotorMapSetNPoints(
  MINSTANCE mInst,
  int motorId,
  int points);
```

LUA Syntax:

```
rc = mc.mcMotorMapSetNPoints(
  number mInst,
  number motorId,
  number points);
```

Description: Set the number of measured points for the screw map of the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	The motor id.
points	An integer specifying the number of measured points.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.
MERROR_INVALID_ARG	**points** is <= 0.

Notes:
Depricated. Use mcMotorMapSetPointCount() instead.

mcMotorMapSetPoint

C/C++ Syntax:

```
int mcMotorMapSetPoint(
  MINSTANCE mInst,
  int motorId,
  int point,
  int error);
```

LUA Syntax:

```
rc = mc.mcMotorMapSetPoint(
  number mInst,
  number motorId,
  number point,
  number error);
```

Description: Set the error (in counts) for the screw map point for the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	The motor id.
point	An integer specifying the measurement point number.
error	An integer specifying the screw error.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
None

Usage:

```
// Set the error in motor 0 screw map.
MINSTANCE mInst = 0;
int rc = mcMotorMapSetPoint(mInst, 0, 5, 12);
```

mcMotorMapSetPointCount

C/C++ Syntax:

```
int mcMotorMapSetPointCount(
  MINSTANCE mInst,
  int motorId,
  int points);
```

LUA Syntax:

```
rc = mc.mcMotorMapSetPointCount(
  number mInst,
  number motorId,
  number points);
```

Description: Set the number of measured points for the screw map for the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	The motor id.
points	An integer specifying the number of measured points.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.
MERROR_INVALID_ARG	**points** is <= 0.

Notes:
None.

Usage:

```
// Set the number of measured points in the screw map for motor 0.
MINSTANCE mInst = 0;
```

```
int rc = mcMotorMapSetPointCount(mInst, 0, 20);
```

mcMotorMapSetStart

C/C++ Syntax:

```
int mcMotorMapSetStart(
  MINSTANCE mInst,
  int motorId,
  int start);
```

LUA Syntax:

```
rc = mc.mcMotorMapSetStart(
  number mInst,
  number motorId,
  number start)
```

Description: Set the starting point number of the screw map of the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	The motor id.
start	An integer specifying the starting point number.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
None/

Usage:

```
// Set the starting point number for the motor 0 screw map. (usually 0)
MINSTANCE mInst = 0;
int rc = mcMotorMapSetStart(mInst, 0, 0);
```

mcMotorRegister

C/C++ Syntax:

```
mcMotorRegister(
  MINSTANCE mInst,
  int motorId);
```

LUA Syntax:

```
N/A
```

Description: Register a motor to the system (used by motion plugins).

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	The motor id.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**motorId** is out of range.

Notes:
Add motor to the core from a motion device. Motion devices can register max of MC_MAX_MOTORS.

Usage:

```
MINSTANCE mInst = 0;
for (int m = MOTOR0; m < MOTOR4; m++) {
    mcMotorRegister(mInst, m);// Register motor0 - motor2 in the core.
}
```

mcMotorSetHomePos

C/C++ Syntax:

```
int mcMotorSetHomePos(
  MINSTANCE mInst,
  int motorId,
  int count);
```

LUA Syntax:

```
rc = mc.mcMotorSetHomePos(
  number mInst,
  number motorId,
  number count)
```

Description: Set the home position for the motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	The motor id.
count	The positon of the motor away from the home switch

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
Sets the position of the motor away from the home switch. This is used to provide a home offest for the motor.

Usage:

```
MINSTANCE mInst = 0;
int m = MOTOR2;
```

```
int count = 1200;
mcMotorSetHomePos(mInst, m, count); // Set the Home position of motor2.
```

mcMotorSetInfoStruct

C/C++ Syntax:

```
int mcMotorSetInfoStruct(
  MINSTANCE mInst,
  int motoId,
  motorinfo_t *minf);
```

LUA Syntax:

```
N/A
```

Description: Retrieve the motor info struct to get all the settings of the motor in one call.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	The motor id.
minf	The address of a motorinfo_t to receive the motor settings.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.
MERROR_INVALID_ARG	**minf** cannot be NULL.

Notes:
To get cuttent settings mcMotorGetInfoStruct() must be called.

```
struct motorinfo {
 double CountsPerUnit;    // Number of encoder counts or steps per unit.
 double MaxVelocity;      // Max velocity of the axis.
 double MaxAcceleration;  // Max rate to accelerate.
 BOOL   Reverse;          // Is the axis reversed?
 double BacklashAmount;   // The amount of backlash in counts.
```

```
 double CurrentVelocity;  // The speed the axis is moving, This could be reporte
 int    CurrentPosition;  // The Current Position (From the motion device).
 BOOL   Homed;            // True if the axis has been homed.
 long   SoftMaxLimit;     // Count for the max travel.
 long   SoftMinLimit;     // Count for the min travel.
 BOOL   CanHome;          // Can this motor home?
 BOOL   Enabled;          // Is this motor enabled?
 long   EnableDelay;      // ms to delay the enable signal for this motor.
};
typedef struct motorinfo motorinfo_t;
```

Usage:

```
MINSTANCE mInst = 0;
int m = MOTOR2;
motorinfo_t minf;
mcMotorGetInfoStruct(mInst, m, &minf;); // Get data for motor2.
minf.CountsPerUnit /= 2;//Divid motor counts per unit by 2.
mcMotorSetInfoStruct(mInst, m, &minf;); // Set data for motor2.
```

mcMotorSetMaxAccel

C/C++ Syntax:

```
int mcMotorSetMaxAccel(
  MINSTANCE mInst,
  int motorId,
  double maxAccel);
```

LUA Syntax:

```
rc = mc.mcMotorSetMaxAccel(
  number mInst,
  number motorId,
  number maxAccel)
```

Description: Set the maximum acceleration value for the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	An integer specifying the motor.
maxAccel	A double specifying the maximum acceleration value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
None.

Usage:

```
// Set a new accel value for motor 0.
MINSTANCE mInst = 0;
int mcMotorSetMaxAccel(mInst, 0, 75);
```

mcMotorSetMaxVel

C/C++ Syntax:

```
int mcMotorSetMaxVel(
  MINSTANCE mInst,
  int motorId,
  double maxVel);
```

LUA Syntax:

```
rc = mc.mcMotorSetMaxVel(
  number mInst,
  number motorId,
  number maxVel)
```

Description: Set the maximum velocity for the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	An integer specifying the motor.
maxVel	A double specifying the maximum velocity value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
None.

Usage:

```
// Set the maximum velocity for motor 0.
MINSTANCE mInst = 0;
int rc = mcMotorSetMaxVel(mInst, 0, 500);
```

mcMotorUnregister

C/C++ Syntax:

```
int mcMotorUnregister(
  MINSTANCE mInst,
  int motorId);
```

LUA Syntax:

```
N/A
```

Description: Unregister a motor from the core (usually not used.)

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	The motor id.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by **motorId** was not found.

Notes:
Remove motor from the core, this should only be done by the owner of the motor (Motion plugin).

Usage:

```
MINSTANCE mInst = 0;
int m = MOTOR2;
mcMotorUnregister(mInst, m);// Unregister motor2 from the Core.
```

mcMpgGetAccel

C/C++ Syntax:

```
int mcMpgGetAccel(
  MINSTANCE mInst,
  int mpg,
  double *percentMaxAccel);
```

LUA Syntax:

```
percentMaxAccel, rc = mc.mcMpgGetAccel(
  number mInst,
  number mpg)
```

Description: Get the percentage of the maximum acceleration for the given MPG.

Parameters:

Parameter	Description
mInst	The controller instance.
mpg	An integer specifying the MPG.
percentMaxAccel	The address of a double to receive the max acceleration percentage.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**percentMaxAccel** is NULL or **mpg** is out of range.

Notes:
None.

Usage:

```
// Get the current accel for MPG 0.
MINSTANCE mInst = 0;
double percentMaxAccel = 0;
int rc;
rc = mcMpgGetAccel(mInst, 0, &percentMaxAccel;);
```

mcMpgGetAxis

C/C++ Syntax:

```
int mcMpgGetAxis(
  MINSTANCE mInst,
  int mpg,
  int *axisId);
```

LUA Syntax:

```
axisId, rc = mc.mcMpgGetAxis(
  number mInst,
  number mpg)
```

Description: Retrieve the current axis for the specified MPG.

Parameters:

Parameter	Description
mInst	The controller instance.
mpg	An integer specifying the MPG.
axisId	the address of a integer to revceive the axis ID.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**axisId** is NULL or **mpg** is out of range.

Notes:
None.

Usage:

```
// Get the currently selected axis for MPG 0.
MINSTANCE mInst = 0;
int axisId;
int rc = mcMpgGetAxis(mInst, 0, &axisId;);
```

mcMpgGetCountsPerDetent

C/C++ Syntax:

```
int mcMpgGetCountsPerDetent(
  MINSTANCE mInst,
  int mpg,
  int *pulses);
```

LUA Syntax:

```
pulses, rc = mc.mcMpgGetCountsPerDetent(
  number mInst,
  number mpg)
```

Description: Retrieves the number of pulse required to move 1 unit for the given MPG.

Parameters:

Parameter	Description
mInst	The controller instance.
mpg	An integer specifying the MPG.
pulses	The address of an integer to receive the pulse count.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**pulses** is NULL or **mpg** is out of range.

Notes:
None.

Usage:

```
// Get the counts per detent for MPG 0.
MINSTANCE mInst = 0;
int cnts;
int rc = mcMpgGetCountsPerDetent(mInst, 0, &cnts;);
```

mcMpgGetEncoderReg

C/C++ Syntax:

```
int mcMpgGetEncoderReg(
  MINSTANCE mInst,
  int mpg,
  HMCREG *hReg);
```

LUA Syntax:

```
hReg, rc = mc.mcMpgGetEncoderReg(
  number mInst,
  number mpg)
```

Description: Get the current encoder register that is associated to the given MPG.

Parameters:

Parameter	Description
mInst	The controller instance.
mpg	An integer specifying the MPG.
hReg	The address of a MCHREG to receive the encoder register's handle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**hReg** is NULL or **mpg** is out of range.

Notes:
None.

Usage:

```
// Get the current encoder associated with MPG 0.
MINSTANCE mInst = 0;
HMCREG hReg = 0;

int rc = mcMpgGetEncoderReg(mInst, 0, &hReg;);
```

mcMpgGetInc

C/C++ Syntax:

```
int mcMpgGetInc(
  MINSTANCE mInst,
  int mpg,
  double *inc);
```

LUA Syntax:

```
inc, rc = mc.mcMpgGetInc(
  number mInst,
  number mpg)
```

Description: Retrieve the increment for the given MPG.

Parameters:

Parameter	Description
mInst	The controller instance.
mpg	An integer specifying the MPG.
inc	The address of a double to receive the increment value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**inc** is NULL or **mpg** is out of range.

Notes:
None.

Usage:

```
// Get the incremnt for MPG 0.
MINSTANCE mInst = 0;
double inc;
int rc = mcMpgGetInc(mInst, 0, &inc;);
```

mcMpgGetRate

C/C++ Syntax:

```
int mcMpgGetRate(
  MINSTANCE mInst,
  int mpg,
  double *percentMaxVel);
```

LUA Syntax:

```
percentMaxVel, rc = mc.mcMpgGetRate(
  number mInst,
  number mpg)
```

Description: REtrieve the percentage of the maximum velocity for the given MPG.

Parameters:

Parameter	Description
mInst	The controller instance.
mpg	An integer specifying the MPG.
percentMaxVel	The address of a double to receive the percentage of max velocity.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**rate** is NULL or **mpg** is out of range.

Notes:
None.

Usage:

```
// Get the velocity percentage for MPG 0.
MINSTANCE mInst = 0;
double velPercent;
int rc = mcMpgGetRate(mInst, 0, &velPercent;);
```

mcMpgGetShuttleMode

C/C++ Syntax:

```
int mcMpgGetShuttleMode(
  MINSTANCE mInst,
  BOOL *on);
```

LUA Syntax:

```
on, rc = mc.mcMpgGetShuttleMode(
  number mInst)
```

Description: Determine the state of MPG shuttle mode.

Parameters:

Parameter	Description
mInst	The controller instance.
on	The address of a BOOL to receive the state of MPG shuttle mode.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**on** is NULL.

Notes:
None.

Usage:

```
// Check the state of MPG shuttle mode.
MINSTANCE mInst = 0;
BOOL enabled = FALSE;
int rc = mcMpgGetShuttleMode(mInst, &enabled;);
```

mcMpgMoveCounts

C/C++ Syntax:

```
int mcMpgMoveCounts(
  MINSTANCE mInst,
  int mpg,
  int deltaCounts);
```

LUA Syntax:

```
rc = mc.mcMpgMoveCounts(
  number mInst,
  number mpg,
  number deltaCounts)
```

Description: Move the axis mapped to the given MPG.

Parameters:

Parameter	Description
mInst	The controller instance.
mpg	An integer specifying the MPG.
deltaCounts	An integer specifying the number of counts the MPG has moved.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**mpg** is out of range.

Notes:
This is the main MPG function that a plugin that supports encoders will use. All the plugin has to do is report the delta difference between reads of the encoder.

Usage:

```
// Move the MPG 0 "n" counts.
MINSTANCE mInst = 0;
int n = 16;
int rc = mcMpgMoveCounts(mInst, 0, n);
```

mcMpgSetAccel

C/C++ Syntax:

```
int mcMpgSetAccel(
  MINSTANCE mInst,
  int mpg,
  double percentMaxAccel);
```

LUA Syntax:

```
rc = mc.mcMpgSetAccel(
  number mInst,
  number mpg,
  number percentMaxAccel)
```

Description: Set the acceration percentage for the given MPG.

Parameters:

Parameter	Description
mInst	The controller instance.
mpg	An integer specifying the MPG.
percentMaxAccel	A double specifying the accelration percentage.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**mpg** is out of range.

Notes:
The accelation percentage is based off of the maximum acceleration values for the currently mapped axis.
The percentage is expressed as a whole percentage. e.g. 20.5 for 20 and a half percent.

Usage:

```
// Set the acceleration value for MPG 0.
MINSTANCE mInst = 0;
int rc = mcMpgSetAccel(mInst, 0, 20.5);
```

mcMpgSetAxis

C/C++ Syntax:

```
int mcMpgSetAxis(
  MINSTANCE mInst,
  int mpg,
  int axisId);
```

LUA Syntax:

```
rc = mc.mcMpgSetAxis(
  number mInst,
  number mpg,
  number axis)
```

Description: Set the current driven axis for the given MPG.

Parameters:

Parameter	Description
mInst	The controller instance.
mpg	An integer specifying the MPG.
axisId	An integer specifying the axis.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.
MERROR_INVALID_ARG	**mpg** is out of range.

Notes:
None.

Usage:

```
// Set MPG 0 to move the X axis.
MINSTANCE mInst = 0;
int rc = mcMpgSetAxis(mInst, 0, X_AXIS);
```

mcMpgSetCountsPerDetent

C/C++ Syntax:

```
int mcMpgSetCountsPerDetent(
  MINSTANCE mInst,
  int mpg,
  int pulses);
```

LUA Syntax:

```
rc = mc.mcMpgSetCountsPerDetent(
  number mInst,
  number mpg,
  number pulses);
```

Description: Set the number of counts the MPG outputs for 1 detent. (usually 4)

Parameters:

Parameter	Description
mInst	The controller instance.
mpg	An integer specifying the MPG.
pulses	An integer specifying the number of counts per detent.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**mpg** is out of range.

Notes:
None.

Usage:

```
// Set the number of counts per detent for MPG 0.
MINSTANCE mInst = 0;
int rc = mcMpgSetCountsPerDetent(mInst, 0, 4);
```

mcMpgSetEncoderReg

C/C++ Syntax:

```
int mcMpgSetEncoderReg(
  MINSTANCE mInst,
  int mpg,
  HMCREG hReg);
```

LUA Syntax:

```
rc = mc.mcMpgSetEncoderReg(
  number mInst,
  number mpg
  number hReg)
```

Description: Associate an encoder register to the given MPG.

Parameters:

Parameter	Description
mInst	The controller instance.
mpg	An integer specifying the MPG.
hReg	A MCHREG to associate to the given MPG.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**mpg** is out of range.

Notes:
None.

Usage:

```
// Associate an encoder with MPG 0.
MINSTANCE mInst = 0;
HMCREG hReg = 0;
int rc
rc = mcRegGetHandle(mInst, "/Sim0/Encoder0", &hReg;);
if (rc == MERROR_NOERROR) {
```

```
 rc = mcMpgSetEncoderReg(mInst, 0, hReg);
}
```

mcMpgSetInc

C/C++ Syntax:

```
int mcMpgSetInc(
  MINSTANCE mInst,
  int mpg,
  double inc);
```

LUA Syntax:

```
rc = mc.mcMpgSetInc(
  number mInst,
  number mpg,
  number inc);
```

Description: Set the desired incremnt (.100, .010, .001, etc...) for the given MPG.

Parameters:

Parameter	Description
mInst	The controller instance.
mpg	An integer specifying the MPG.
inc	A double specifying the increment.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**mpg** is out of range.

Notes:
None.

Usage:

```
// Set the increment to .001 for MPG 0.
MINSTANCE mInst = 0;
int rc = mcMpgSetInc(mInst, 0, .001);
```

mcMpgSetRate

C/C++ Syntax:

```
int mcMpgSetRate(
  MINSTANCE mInst,
  int mpg,
  double percentMaxVel);
```

LUA Syntax:

```
rc = mc.mcMpgSetRate(
  number mInst,
  number mpg,
  number percentMaxVel)
```

Description: Set the percentage of the maximum velocity for the given MPG.

Parameters:

Parameter	Description
mInst	The controller instance.
mpg	An integer specifying the MPG.
percentMaxVel	A double specifying the percentage of the maximum velocity.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**mpg** is out of range.

Notes:
The velocity percentage is based off of the maximum velocity value for the currently mapped axis. The percentage is expressed as a whole percentage. e.g. 20.5 for 20 and a half percent.

Usage:

```
// Set the rate for MPG 0 to 25.0%
MINSTANCE mInst = 0;
int rc = mcMpgSetRate(mInst, 0, 25.0);
```

mcMpgSetShuttleMode

C/C++ Syntax:

```
int mcMpgSetShuttleMode(
  MINSTANCE mInst,
  BOOL on);
```

LUA Syntax:

```
rc = mc.mcMpgSetShuttleMode(
  number mInst,
  number on);
```

Description: Set the state of MPG shuttle mode.

Parameters:

Parameter	Description
mInst	The controller instance.
on	A BOOL specifying the state of the MPG shuttle mode.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Turn on MPG shuttle mode.
MINSTANCE mInst = 0;
int rc = mcMpgSetShuttleMode(mInst, MC_ON);
```

mcPluginConfigure

C/C++ Syntax:

```
int mcPluginConfigure(
  MINSTANCE mInst,
  int plugId);
```

LUA Syntax:

```
N/A
```

Description: Display the plugin configuration dialog for the given plugin ID.

Parameters:

Parameter	Description
mInst	The controller instance.
plugId	An integer specifying the plugin ID.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_PLUGIN_NOT_FOUND	The plugin specified by **plugId** was not found.

Notes:
Plugins can contain multiple devices. However, only one configuration dialog is used so plugins must have a dialog that is capable of configuring every device that belongs to it (if needed).

Usage:

```
MINSTANCE mInst = 0;
HMCPLUG hPlug = 0;
pluginfo_t pluginf;

// Loop through all of the plugins looking for plugins
// that have configuration dialogs.
while (mcPluginGetNextHandle(PLUG_TYPE_CONFIG, hPlug, &hPlug;) == MERROR_NOERROR)
 // Get the plugin info that contains the plugin ID.
```

```
 rc = mcPluginGetInfoStruct(hPlug, &pluginf;);
 if (rc == MERROR_NOERROR) {
  int pluginId = pluginf.plugId;
  // Call the configuration dialog.
  rc = mcPluginConfigure(mInst, pluginId);;
 }
}
```

mcPluginCoreLoad

C/C++ Syntax:

```
int mcPluginCoreLoad(
  const char *shortName);
```

LUA Syntax:

```
N/A
```

Description: Load a plugin into the core.

Parameters:

Parameter	Description
shortName	A string buffer specifying the base name of the plugin.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_PLUGIN_NOT_FOUND	The plugin specified by **shortName** was not found.

Notes:
Normally, all plugins are found and loaded by the core at stratup. This function provides a means to load a plugin after startup. For instance, if the plugin was installed after startup. The function uses the base name of the plugin to locate it by appending the platform specific extension (**m4pw** for Windows, **m4pl** for Linux, and **m4pm** for Mac).

Usage:

```
// Load the mcGalil plugin.
int rc = mcPluginCoreLoad("mcGalil");
```

mcPluginCoreUnload

C/C++ Syntax:

```
int mcPluginCoreUnload(
  const char *shortName);
```

LUA Syntax:

```
N/A
```

Description: Unload a plugin from the core.

Parameters:

Parameter	Description
shortName	A string buffer specifying the base name of the plugin.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_PLUGIN_NOT_FOUND	The plugin specified by **shortName** was not found.

Notes:
About the only reason to unload a plugin from the core is if it is being uninstalled. Plugins should be tested during their development to ensure that they can be unloaded safely without causing crashes or other abnormal behavior.

Usage:

```
// Unload the mcGalil plugin.
int rc = mcPluginCoreUnload("mcGalil");
```

mcPluginDiagnostic

C/C++ Syntax:

```
int mcPluginDiagnostic(
  MINSTANCE mInst,
  int pluginId);
```

LUA Syntax:

```
N/A
```

Description: Display the plugin diagnostic dialog for the given plugin ID.

Parameters:

Parameter	Description
mInst	The controller instance.
plugId	An integer specifying the plugin ID.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_PLUGIN_NOT_FOUND	The plugin specified by **plugId** was not found.

Notes:
Plugins can contain multiple devices. However, only one diagnostic dialog is used so plugins must have a dialog that is capable of diagnosing every device that belongs to it (if needed).

Usage:

```
MINSTANCE mInst = 0;
HMCPLUG hPlug = 0;
pluginfo_t pluginf;

// Loop through all of the plugins looking for plugins
// that have diagnostic dialogs.
while (mcPluginGetNextHandle(PLUG_TYPE_DIAG, hPlug, &hPlug;) == MERROR_NOERROR)
 // Get the plugin info that contains the plugin ID.
```

```
 rc = mcPluginGetInfoStruct(hPlug, &pluginf;);
 if (rc == MERROR_NOERROR) {
  int pluginId = pluginf.plugId;
  // Call the diagnostics dialog.
  rc = mcPluginDiagnostic(mInst, pluginId);;
 }
}
```

mcPluginEnable

C/C++ Syntax:

```
int mcPluginEnable(
  HMCPLUG hPlug,
  BOOL enable);
```

LUA Syntax:

```
N/A
```

Description: Enable a plugin.

Parameters:

Parameter	Description
hPlug	A HMCPLUG handle specifying the plugin.
enable	A BOOL specifying the plugin enable state.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_PLUGIN_NOT_FOUND	The plugin specified by **hPlug** was not found.

Notes:
None.

Usage:

```
HMCPLUG hPlug = 0;
// Loop through all of the plugins looking for plugins
// that have configuration dialogs.
while (mcPluginGetNextHandle(PLUG_TYPE_CONFIG, hPlug, &hPlug;) == MERROR_NOERROR)
 // Enable the plugin.
 rc = mcPluginEnable(hPlug, TRUE);
}
```

mcPluginGetEnabled

C/C++ Syntax:

```
int mcPluginGetEnabled(
  HMCPLUG hPlug,
  BOOL *enabled);
```

LUA Syntax:

```
N/A
```

Description:

Parameters:

Parameter	Description
hPlug	A HMCPLUG handle specifying the plugin.
enabled	The address of a BOOL to receive plugin enable state.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_PLUGIN_NOT_FOUND	The plugin specified by **hPlug** was not found.
MERROR_INVALID_ARG	**enabled** is NULL.

Notes:

Usage:

```
HMCPLUG hPlug = 0;
BOOL enabled = FALSE;
// Loop through all of the plugins looking for plugins
// that have configuration dialogs.
while (mcPluginGetNextHandle(PLUG_TYPE_CONFIG, hPlug, &hPlug;) == MERROR_NOERROR)
 // See if the plugin is enabled..
 rc = mcPluginGetEnabled(hPlug, &enabled;);
 if (rc == MERROR_NOERROR) {
  if (enabled == TRUE) {
   // The plugin is enabled!!!
  } else {
   // The plugin is not enabled!!!
```

```
    }
  }
}
```

mcPluginGetInfoStruct

C/C++ Syntax:

```
mcPluginGetInfoStruct(HMCPLUG hPlug, pluginfo_t *pluginf);
```

LUA Syntax:

```
N/A
```

Description:

Parameters:

Parameter	Description
hPlug	A HMCPLUG handle specifying the plugin.
pluginf	The address of a pluginfo_t struct to receive the plugin information.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_PLUGIN_NOT_FOUND	The plugin specified by **hPlug** was not found.
MERROR_INVALID_ARG	**pluginf** is NULL.

Notes:

Usage:

```
HMCPLUG hPlug = 0;
pluginfo_t pluginf;

// Loop through all of the plugins.
while (mcPluginGetNextHandle(PLUG_TYPE_ALL, hPlug, &hPlug;) == MERROR_NOERROR) {
 // Get the plugin info that contains the plugin ID.
 rc = mcPluginGetInfoStruct(hPlug, &pluginf;);
 if (rc == MERROR_NOERROR) {
  int pluginId = pluginf.plugId;
  ...
 }
}
```

mcPluginGetLicenseFeature

C/C++ Syntax:

```
int mcPluginGetLicenseFeature(
  HMCPLUG hPlug,
  int index,
  char *buf,
  int bufSize,
  BOOL *status);
```

LUA Syntax:

```
N/A
```

Description: Get the plugin licensed features by the given index and return their status (licensed or not).

Parameters:

Parameter	Description
mInst	The controller instance.
index	An integer specifying the index of the license feature to retrieve.
buf	A string buffer to receive the license feature name.
bufSize	The length of the string buffer.
status	The address of a BOOL to receive the license status of the feature (TRUE/FALSE).

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	**hPlug** is zero, **buf** or **status** is NULL, **bufSize** is <= 0.
MERROR_PLUGIN_NOT_FOUND	The plugin specified by **hPlug** was not found.
MERROR_NODATA	No more licensed features can be found.

Notes:

Usage:

```
HMCPLUG hPlug = 0;
// Loop through all of the plugins.
while (mcPluginGetNextHandle(PLUG_TYPE_ALL, hPlug, &hPlug;) == MERROR_NOERROR) {
 // Get all of the plugin license features
 int index = 0;
 char buf[80];
 BOOL licensed = FALSE;
 while (mcPluginGetLicenseFeature(hPlug, index, buf, sizeof(buf), &licensed;) ==
  printf("license feature %s is %s.\n", buf, licensed == TRUE ? "licensed" : "not
  index++;
 }
}
```

mcPluginGetNextHandle

C/C++ Syntax:

```
int mcPluginGetNextHandle(
  int plugType,
  HMCPLUG startPlug,
  HMCPLUG *hPlug);
```

LUA Syntax:

```
N/A
```

Description: Provides a means of looping through all or only plugins with specific attributes.

Parameters:

Parameter	Description
plugType	An integer specifying the plugin attributes. (PLUG_TYPE_NONE, PLUG_TYPE_MOTION, PLUG_TYPE_IO, PLUG_TYPE_SCRIPT, PLUG_TYPE_CONFIG, PLUG_TYPE_DIAG, PLUG_TYPE_REG, PLUG_TYPE_JOG, PLUG_TYPE_ALL)
startPlug	The starting HMCPLUG handle with which to retrieve the next handle in the list. (Must be 0 to retrieve the first handle in the list)
hPlug	The address of a HMCPLUG to receive the next plugin handle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_NODATA	No more plugins with the specified attributes can be found.

Notes:

Usage:

```
HMCPLUG hPlug = 0;
```

```
// Loop through all of the plugins looking for plugins
// that have configuration dialogs and privide I/O.
int attr = PLUG_TYPE_CONFIG | PLUG_TYPE_IO;
while (mcPluginGetNextHandle(attr, hPlug, &hPlug;) == MERROR_NOERROR) {
 // Enable the plugin.
 rc = mcPluginEnable(hPlug, TRUE);
}
```

mcPluginGetValid

C/C++ Syntax:

```
int mcPluginGetValid(
  HMCPLUG hPlug,
  BOOL *valid);
```

LUA Syntax:

```
N/A
```

Description: Check to see if the plugin is valid.

Parameters:

Parameter	Description
hPlug	A HMCPLUG specifying the plugin handle.
valid	The address of a BOOL to receive the plugins validity status.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	**hPlug** is 0 or **valid** is NULL
MERROR_PLUGIN_NOT_FOUND	The plugin specified by **hPlug** cannot be found.

Notes:

Plugin validity is a function of the plugin signature resolving to a certified Mach developer. If no signature file is provided with the plugin or it is invalid, the plugin will not function.

Plugin signatures are generated with a developer key. Developer keys can be requested by contacting Newfangled Solutions, LLC.

Usage:

```
HMCPLUG hPlug = 0;
BOOL valid = FALSE;
// Loop through all of the plugins checking to see if they are valid.
```

```
while (mcPluginGetNextHandle(PLUG_TYPE_ALL, hPlug, &hPlug;) == MERROR_NOERROR) {
 // Check the plugin validity.
 rc = mcPluginGetValid(hPlug, &valid;);
 if (rc == MERROR_NOERROR) {
  if (valid == TRUE) {
   // The plugin is valid and will function.
  } else {
   // The plugin is invalid and will not function.
  }
 }
}
```

mcPluginInstall

C/C++ Syntax:

```
int mcPluginInstall(
  const char *m4plug);
```

LUA Syntax:

```
N/A
```

Description: Provides a means of installing a plugin from a plugin archive.

Parameters:

Parameter	Description
m4Plug	A string buffer specifying the plugin archive to install.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_PLUGIN_NOT_FOUND	The plugin archive specified by **m4Plug** is not valid.

Notes:
Plugin archive name extentions vary by the plafrom. **m4plugw** for Windows, **m4plugl** for Linux, **m4plugm** for Mac. If one were to try and install a windows plugin archive on a Linux machine, then MERROR_PLUGIN_NOT_FOUND would be returned.

Usage:

```
// Install the Windows mcGalil plugin from an installation archive.
int rc = mcPluginInstall("mcGalil.m4Plugw");
if (rc == MERROR_NOERROR) {
 //The plugin installed successfully.
}
```

mcPluginRegister

C/C++ Syntax:

```
int mcPluginRegister(
  HMCPLUG hPlug,
  const char *DeveloperId,
  const char *Desc,
  const char *Version,
  int Type);
```

LUA Syntax:

```
N/A
```

Description: Registers the plugin to the core.

Parameters:

Parameter	Description
hPlug	A HMCPLUG handle (generated and provided by the core) that defines the current plugin.
DeveloperId	A string buffer specifying the developer ID.
Desc	A string buffer specifying the plugins description.
Version	A string buffer specifying the plugins version.
Type	A integer specifying the plugins attributes (or capabilities). It can be the logical **OR** combination of PLUG_TYPE_MOTION, PLUG_TYPE_IO, PLUG_TYPE_SCRIPT, PLUG_TYPE_CONFIG, PLUG_TYPE_DIAG, PLUG_TYPE_REG, and PLUG_TYPE_JOG.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_PLUGIN_NOT_FOUND	The plugin specified by **hPlug** is not valid.

Notes:
Once a plugin is found by the core, the core loads the plugin and calls its mcPluginLoad() entry point

function. The plugin should then register itself with the core at this time.

Usage:

```
// This function gets called (only once) when the Plugin is loaded by Mach Core.
MCP_API int MCP_APIENTRY mcPluginLoad(HMCPLUG id)
{
 ...
 mcPluginRegister(id, "Newfangled",
  "Simulator - Newfangled Solutions", MC_VERSION_STR,
  PLUG_TYPE_MOTION | PLUG_TYPE_IO  | PLUG_TYPE_REG | PLUG_TYPE_CONFIG | PLUG_TYPE
 return(MERROR_NOERROR);
}
```

mcPluginRemove

C/C++ Syntax:

```
int mcPluginRemove(const char *shortName);
```

LUA Syntax:

```
N/A
```

Description: Unloads a plugin and then rmoves the plugin from the core's Plugins directory.

Parameters:

Parameter	Description
shortName	A string buffer specifying the short name (base name).

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
If the plugin sepcified by **shortName** was installed using a plugin archive, then all of the files installed by said archive are removed. Otherwise, only the plugin itself is removed from the Plugins directory.

Usage:

```
// Remove the mcGalil plugin.
MINSTANCE mInst = 0;
int rc = mcPluginRemove("mcGalil");
```

mcProfileDeleteKey

C/C++ Syntax:

```
int mcProfileDeleteKey(
  MINSTANCE mInst,
  const char *section,
  const char *key);
```

LUA Syntax:

```
rc = mc.mcProfileDeleteKey(
  number mInst,
  string section,
  string key)
```

Description: Delete key from the profile's INI file.

Parameters:

Parameter	Description
mInst	The controller instance.
section	A pointer to a char buffer with section string.
key	A pointer to a char buffer with key string.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	One or more of the parameters had a NULL value.

Notes:
None.

Usage:

```
MINSTANCE mInst = 0;
mcProfileDeleteKey(mInst , "P_Port", "Frequency");
```

mcProfileDeleteSection

C/C++ Syntax:

```
int mcProfileDeleteSection(
  MINSTANCE mInst,
  const char *section);
```

LUA Syntax:

```
rc = mc.mcProfileDeleteSection(
  number mInst,
  string section)
```

Description: Delete the specified section from the profile's INI file.

Parameters:

Parameter	Description
mInst	The controller instance.
section	A string buffer buffer specifying the section.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	One or more of the parameters had a NULL value.

Notes:
None.

Usage:

```
MINSTANCE mInst = 0;
mcProfileDeleteKey(mInst , "P_Port");
```

mcProfileExists

C/C++ Syntax:

```
int mcProfileExists(
  MINSTANCE mInst,
  const char *section,
  const char *key);
```

LUA Syntax:

```
rc = mc.mcProfileExists(
  number mInst,
  string section,
  string key)
```

Description: Determine if the section or section and key are in the profiles INI file.

Parameters:

Parameter	Description
mInst	The controller instance.
section	A string buffer specifying the section.
key	A string buffer specifying the key.

Returns:

Return Code	Description
MC_TRUE	The item pecified by **section** and **key** was found in INI file.
MC_FALSE	The item pecified by **section** and **key** was not found in INI file.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**section** cannot be NULL.

Notes:
If you wish to check only for the existence of a section, use NULL for the value of **key**.

Usage:

```
MINSTANCE mInst = 0;
mcProfileWriteInt(mInst, "P_Port", "Frequency");
```

mcProfileGetDouble

C/C++ Syntax:

```
mcProfileGetDouble(
  MINSTANCE mInst,
  const char *section,
  const char *key,
  double *retval,
  double defval);
```

LUA Syntax:

```
retval, rc = mc.mcProfileGetDouble(
  number mInst,
  string section,
  string key,
  number defval)
```

Description: Read a double value from the profile's INI file.

Parameters:

Parameter	Description
mInst	The controller instance.
section	A string buffer specifying the section.
key	A string buffer specifying the key.
retval	The address of a double to receive the value from the section and key location.
defval	The default value if the item is not found.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	One or more of the parameters had a NULL value.

Notes:
Reading the profile settings can be done at any time. If **section** and **key** are not in the file, they are created and assigned the default value.

Usage:

```
double rval=0;
MINSTANCE mInst = 0;
mcProfileGetDouble(mInst , "P_Port", "Frequency", &rval;, 25.34);
```

mcProfileGetInt

C/C++ Syntax:

```
int mcProfileGetInt(
  MINSTANCE mInst,
  const char *section,
  const char *key,
  long *retval,
  long defval);
```

LUA Syntax:

```
retval, rc = mc.mcProfileGetInt(
  number mInst,
  string section,
  string key,
  number defval)
```

Description: Read a long value from the profile's INI file.

Parameters:

Parameter	Description
mInst	The controller instance.
section	A string buffer specifying the section.
key	A string buffer specifying the key.
retval	The address of a long to receive the value from the section and key location.
defval	The default value if the item is not found.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
No Error.	
MERROR_INVALID_ARG	One or more of the parameters had a NULL value.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
The mInst parameter was out of	

range.

Notes:
Reading the profile settings can be done at any time. If **section** and **key** are not in the file, they are created and assigned the default value.

Usage:

```
long rval=0;
int mInst=0;
mcProfileGetInt(mInst , "P_Port", "Frequency", &rval;, 25000);
```

mcProfileGetName

C/C++ Syntax:

```
int mcProfileGetName(
  MINSTANCE mInst,
  char *buff,
  size_t bufsize);
```

LUA Syntax:

```
name, rc = mc.mcProfileGetName(
  number mInst)
```

Description: Retrieve the current profile name.

Parameters:

Parameter	Description
mInst	The controller instance.
buff	A string buffer to receive the profile's name.
buffsize	The length of the buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	One or more of the parameters had a NULL value.

Notes:
None

Usage:

```
MINSTANCE mInst = 0;
char buff[80];
memset(buff, 0, 80);
mcProfileGetName(mInst, buff, 80);
```

mcProfileGetString

C/C++ Syntax:

```
int mcProfileGetString(
  MINSTANCE mInst,
  const char *section,
  const char *key,
  char *buff,
  long buffsize,
  const char *defval);
```

LUA Syntax:

```
buff, rc = mc.mcProfileGetString(
  MINSTANCE mInst,
  string section,
  string key,
  string defval);
```

Description: Read a string value from the profile's INI file.

Parameters:

Parameter	Description
mInst	The controller instance.
section	A string buffer specifying the section.
key	A string buffer specifying the key.
buff	A string buffer to receive the string from the section and key location.
buffsize	The length of the buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	One or more of the parameters had a NULL value.

Notes:
Reading the profile settings can be done at any time. If **section** and **key** are not in the file, they are

created and assigned the default value.

Usage:

```
MINSTANCE m_inst = 0;
char *key = "BufferedTime";
char buff[80];
memset(buff, 0, 80);
mcProfileGetString(mInst , "SomeSection", key, buff, 80, "0.100");
```

mcProfileSave

C/C++ Syntax:

```
int mcProfileSave(
  INSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcProfileSave(
  number mInst)
```

Description: Save the profile's settings to the INI file.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Saving the profile will write the settings to the profile's INI file. This will ensure that a power failue will not result in lost settings. The settings that are saved are the core settings. Plugin and GUI settings fall under the responsability of the GUI or plugin programmer.

Usage:

```
MINSTANCE mInst = 0;
mcProfileSave(mInst); // Flush the settings to the INI file.
```

mcProfileWriteDouble

C/C++ Syntax:

```
int mcProfileWriteDouble(
  MINSTANCE mInst,
  const char *section,
  const char *key,
  double val);
```

LUA Syntax:

```
rc = mc.mcProfileWriteDouble(
  number mInst,
  string section,
  string key,
  double val)
```

Description: Write a double value to the profile's INI file.

Parameters:

Parameter	Description
mInst	The controller instance.
section	A string buffer specifying the section.
key	A string buffer specifying the key.
val	A double specifying the value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	One or more of the parameters had a NULL value.

Notes:
Writing to the profile settings can be done at any time.

Usage:

```
MINSTANCE mInst=0;
```

```
mcProfileWriteDouble(m_cid , "P_Port", "Frequency", 45000);
```

mcProfileWriteInt

C/C++ Syntax:

```
int mcProfileWriteInt(
  MINSTANCE mInst,
  const char *section,
  const char *key,
  long val);
```

LUA Syntax:

```
int mcProfileWriteInt(
  number mInst,
  string section,
  string key,
  number val)
```

Description: Write a long value to the profile's INI file.

Parameters:

Parameter	Description
mInst	The controller instance.
section	A string buffer specifying the section.
key	A string buffer specifying the key.
val	A long specifying the value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	One or more of the parameters had a NULL value.

Notes:
Writing to the profile settings can be done at any time.

Usage:

```
MINSTANCE mInst = 0;
```

```
mcProfileWriteInt(m_cid , "P_Port", "Frequency", 45000);
```

mcProfileWriteString

C/C++ Syntax:

```
int mcProfileWriteString(
  MINSTANCE mInst,
  const char *section,
  const char *key,
  const char *val);
```

LUA Syntax:

```
rc = mc.mcProfileWriteString(
  number mInst,
  string section,
  string key,
  string val)
```

Description: Write a string value to the profile's INI file.

Parameters:

Parameter	Description
mInst	The controller instance.
section	A string buffer specifying the section.
key	A string buffer specifying the key.
val	A string buffer specifying the value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	One or more of the parameters had a NULL value.

Notes:
Writing to the profile settings can be done at any time.

Usage:

```
MINSTANCE m_inst = 0;
```

```
char *key = "BufferedTime"
double BuffTime = .250;
char val[80];
sprintf(val, "%.4f", BuffTime);
mcProfileWriteString(mInst, "Darwin", key, val);
```

mcRegGetCommand

C/C++ Syntax:

```
int mcRegGetCommand(
  HMCREG hReg,
  char *cmd,
  size_t cmdLen)
```

LUA Syntax:

```
N/A
```

Description: Retrieve a command from the given register.

Parameters:

Parameter	Description
hReg	A HMCREG specifying the register handle.
cmd	A string buffer to receive the command.
cmdLen	The sized of the command buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_REG_NOT_FOUND	The **hReg** parameter was 0 or invlaid.
MERROR_INVALID_ARG	**cmd** is NULL.
MERROR_INVALID_TYPE	The register is not of type **REG_TYPE_COMMAND**.

Notes:
The function will not return a command larger than 1024 bytes.

See Also:

- mcRegSendCommand

and

- mcRegSetResponse

Usage:

```
// Retrieve the register command.
int simControl::ProcessMsg(long msg, long param1, long param2)
{
 HMCREG hReg = (HMCREG)param1;
 long RegVal;
 switch (msg) {
 case MSG_REG_COMMAND:
  mcRegGetValueLong(hReg, &RegVal;);
  if (hReg = m_RegCommand) { // Is this our command register?
   char command[1024];
   mcRegGetCommand(hReg, command, sizeof(command));
   wxString cmd(command);
   cmd.MakeUpper();
   if (cmd == wxT("THC ON")) {
    m_thc = true;
    mcRegSetResponse(hReg, "OK");
   } else if (cmd == wxT("THC OFF")) {
    m_thc = false;
    mcMotionSync(m_cid);
    mcRegSetResponse(hReg, "OK");
   } else if (cmd == wxT("THC STATUS")) {
    if (m_thc) {
     mcRegSetResponse(hReg, "1");
    } else {
     mcRegSetResponse(hReg, "0");
    }
   }
  }
  break;
 default:
  ;
 }
 return(MERROR_NOERROR);
}
```

mcRegGetHandle

C/C++ Syntax:

```
int mcRegGetHandle(
  MINSTANCE mInst,
  const char *path,
  HMCREG *hReg);
```

LUA Syntax:

```
hReg, rc = mc.mcRegGetHandle(
  number mInst,
  string path)
```

Description: Retrieve a handle to a register give its path.

Parameters:

Parameter	Description
mInst	The controller instance.
path	A string buffer specifying the register path.
hReg	The address of a HMCREG to receive the register handle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**path** or **hReg** is NULL.

Notes:
None.

Usage:

```
// Get the register handle.
MINSTANCE mInst = 0;
HMCREG hReg;
int rc;
double value;
if (mcRegGetHandle(mInst, "core/global/Version", &hReg;) == MERROR_NOERROR) {
```

```
 rc = mcRegGetValue(hReg, &value;);
}
```

mcRegGetInfo

C/C++ Syntax:

```
int mcRegGetInfo(
  HMCREG hReg,
  char *nameBuf,
  size_t nameBuflen,
  char *descBuf,
  size_t descBuflen,
  int *type,
  HMCDEV *hDev);
```

LUA Syntax:

```
nameBuf, descBuf, type, hDev, rc = mc.mcRegGetInfo(
  number hReg)
```

Description: Return infromation about a register.

Parameters:

Parameter	Description
hReg	A HMCREG specifying the register handle.
nameBuf	A string buffer to receive the name of the register.
nameBuflen	The length of the name buffer.
descBuf	A string buffer to receive the description of the register.
descBuflen	The length of the description buffer.
type	The address of an integer to receive the register type.
hDev	The address of an integer to receive the register handle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_REG_NOT_FOUND	The **hReg** parameter was 0 or invlaid.

Notes:
The hReg parameter **MUST** be a valid signal handle. Otherwise, the function will crash. **nameBuf** ,**descBuf**, **type**, or **hDev** can be NULL depending on the amout of information required.

Usage:

```
HMCREG hReg = 0;
char name[80];
char desc[80];
int type;
HMCDEV hDev;

while (mcSignalGetNextHandle(hDev, hReg, &hReg;) == MERROR_NOERROR) {
 if (hSig != 0) {
  // Get the info on the register.
  mcRegGetInfo(hReg, name, sizeof(name), desc, sizeof(desc), &type;, &hDev;));
 } else {
  break;
 }
}
```

mcRegGetInfoStruct

C/C++ Syntax:

```
mcRegGetInfoStruct(
  HMCREG hReg,
  reginfo_t *reginf);
```

LUA Syntax:

```
N/A
```

Description: Return infromation about a register.

Parameters:

Parameter	Description
hReg	A HMCREG specifying the register handle.
reginf	The adress of a reginfo struct to receive the register information.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_REG_NOT_FOUND	The **hReg** parameter was 0 or invlaid.
MERROR_INVALID_ARG	**reginf** is NULL

Notes:
The hReg parameter **MUST** be a valid signal handle. Otherwise, the function will crash.

```
struct reginfo {
 char regName[80];
 char regDesc[80];
 int  regType;
 HMCDEV regDev;
 void *regUserData;
 int regInput;
};
typedef struct reginfo reginfo_t;
```

Usage:

```
HMCREG hReg = 0;
reginfo_t reginf;

while (mcSignalGetNextHandle(hDev, hReg, &hReg;) == MERROR_NOERROR) {
 if (hSig != 0) {
  // Get the info on the register.
  mcRegGetInfoStruct(hReg, ®inf;);
 } else {
  break;
 }
}
```

mcRegGetNextHandle

C/C++ Syntax:

```
int mcRegGetNextHandle(
  HMCDEV hDev,
  HMCREG startReg,
  HMCREG *hReg);
```

LUA Syntax:

```
hReg, rc = mc.mcRegGetNextHandle(
  number hDev,
  number startReg)
```

Description: Provides a means of looping through the available registers.

Parameters:

Parameter	Description
hDev	the register's parent device handle.
startReg	The starting register handle.
hReg	The address of a hMCREG to receive the next register handle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_DEVICE_NOT_FOUND	The **hDev** parameter was 0 or invlaid.
MERROR_INVALID_ARG	**hReg** is NULL.
MERROR_NODATA	No more register handles can be found.

Notes:
To start, call mcRegGetNextHandle() with startReg = 0. hDev MUST be valid or the application can crash.

Usage:

```
HMCSIG hReg = 0;
HMCDEV hDev;
mcDeviceGetHandle(0, 0,&hDev;);
while (mcRegisterGetNextHandle(hDev, hReg, &hReg;) == MERROR_NOERROR) {
 if (hSig != 0) {
  // do something with hReg.
 } else {
  break;
 }
}
```

mcRegGetUserData

C/C++ Syntax:

```
int mcRegGetUserData(
  HMCREG hReg,
  void **data);
```

LUA Syntax:

```
N/A
```

Description: Retrieve the user data pointer for the hReg.

Parameters:

Parameter	Description
hReg	A HMCREG specifying the register handle.
data	The address of a void pointer receive the user data pointer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_REG_NOT_FOUND	The **hReg** parameter was 0 or invlaid.
MERROR_INVALID_ARG	**data** is NULL.

Notes:
The hReg parameter **MUST** be a valid register handle. Otherwise, the function will crash.
This function allows the programmer to retreive a pointer to asscoiated private data that is local to the register's parent plugin/device.

Usage:

```
struct myDataStruct {
 int myInt;
 char myString[80];
 ...
};

HMCREG hReg;
int mInst=0;
```

```
myDataStruct *data = NULL;
void *dataPtr;
// Get the handle to holding register 1 on SIM0.
if (mcRegGetHandle(mInst, "SIM0/HoldingReg1", &hReg;) == MERROR_NOERROR) {
 mcRegGetUserData(hReg, &dataPtr;);
 data = (myDataStruct *) dataPtr;
}
```

mcRegGetValue

C/C++ Syntax:

```
int mcRegGetValue(
  HMCREG hReg,
  double *value);
```

LUA Syntax:

```
value, rc = mcRegGetValue(
  number hReg)
```

Description: Retrieve a register's double value.

Parameters:

Parameter	Description
hReg	A HMCREG specifying the register handle.
value	The address of a double to receive the value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_REG_NOT_FOUND	The **hReg** parameter was 0 or invlaid.
MERROR_INVALID_ARG	**value** is NULL.

Notes:
None.

Usage:

```
// Get the register value.
MINSTANCE mInst = 0;
HMCREG hReg;
int rc;
double value;
if (mcRegGetHandle(mInst, "core/global/Version", &hReg;) == MERROR_NOERROR) {
 rc = mcRegGetValue(hReg, &value;);
}
```

mcRegGetValueLong

C/C++ Syntax:

```
int mcRegGetValueLong(
  HMCREG hReg,
  double *value);
```

LUA Syntax:

```
value, rc = mcRegGetValueLong(
  number hReg)
```

Description: Retrieve a register's long value.

Parameters:

Parameter	Description
hReg	A HMCREG specifying the register handle.
value	The address of a long to receive the value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_REG_NOT_FOUND	The **hReg** parameter was 0 or invlaid.
MERROR_INVALID_ARG	**value** is NULL.

Notes:
None.

Usage:

```
// Get the register value.
MINSTANCE mInst = 0;
HMCREG hReg;
int rc;
long value;
if (mcRegGetHandle(mInst, "core/global/Version", &hReg;) == MERROR_NOERROR) {
 rc = mcRegGetValueLong(hReg, &value;);
}
```

mcRegGetValueString

C/C++ Syntax:

```
int mcRegGetValueString(
  HMCREG hReg,
  char *buf,
  size_t bufSize);
```

LUA Syntax:

```
buf, rc = mcRegGetValueString(
  number hReg)
```

Description: Retrieve a register's string value.

Parameters:

Parameter	Description
hReg	A HMCREG specifying the register handle.
buf	A string buffer to receive the value.
bufSize	The length of the string buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_REG_NOT_FOUND	The **hReg** parameter was 0 or invlaid.
MERROR_INVALID_ARG	**buf** is NULL.

Notes:
None.

Usage:

```
// Get the register value.
MINSTANCE mInst = 0;
HMCREG hReg;
int rc;
char value[80];
if (mcRegGetHandle(mInst, "core/global/Version", &hReg;) == MERROR_NOERROR) {
 rc = mcRegGetValueString(hReg, &value;, sizeof(value));
}
```

mcRegGetValueStringClear

C/C++ Syntax:

```
int mcRegGetValueStringClear(
  HMCREG hReg,
  char *buf,
  size_t bufSize);
```

LUA Syntax:

```
buf, rc = mcRegGetValueString(
  number hReg)
```

Description: Retrieve a register's string value and then clear the register.

Parameters:

Parameter	Description
hReg	A HMCREG specifying the register handle.
buf	A string buffer to receive the value.
bufSize	The length of the string buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_REG_NOT_FOUND	The **hReg** parameter was 0 or invlaid.
MERROR_INVALID_ARG	**buf** is NULL.

Notes:
None.

Usage:

```
// Get the register value and clear it.
MINSTANCE mInst = 0;
HMCREG hReg;
int rc;
char value[80];
if (mcRegGetHandle(mInst, "core/global/Version", &hReg;) == MERROR_NOERROR) {
 rc = mcRegGetValueStringClear(hReg, &value;, sizeof(value));
}
```

mcRegRegister

C/C++ Syntax:

```
int mcRegRegister(
  HMCDEV hDev,
  const char *regName,
  const char *regDesc,
  int regType,
  HMCREG *hReg);
```

LUA Syntax:

```
N/A
```

Description: Adds a register to the core.

Parameters:

Parameter	Description
hDev	The handle of the register's parent device.
regName	A string buffer specifying the name of the register.
regDesc	A string buffer specifying the description of the register.
regType	REG_TYPE_NONE, REG_TYPE_INPUT, REG_TYPE_OUTPUT, REG_TYPE_HOLDING, REG_TYPE_COMMAND, REG_TYPE_ENCODER.
hReg	The address of a HMCREG that receives the register handle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_DEVICE_NOT_FOUND	The **hDev** parameter was 0 or invlaid.
MERROR_INVALID_ARG	**regName**, **regDesc**, or **hReg** is NULL.

Notes:
The hDev parameter **MUST** be a valid device handle. Otherwise, the function will crash.

Registers can contain data of any type, even string data. The maximum string length is 4096 bytes for registers other than **REG_TYPE_COMMAND** type registers and 1024 bytes for the **REG_TYPE_COMMAND** type registers.

REG_TYPE_NONE specifies a register that is not typed. This kind of register is basically just a data container that will not notify any process if the data it stores changes.

REG_TYPE_INPUT specifies a register that can be mapped to a Mach input signal. Mach and any GUI front end is notified if the data in this type of register is changed with a **MSG_REG_CHANGED** message. This type of register usually contains boolean data.

REG_TYPE_OUTPUT specifies a register that can be mapped to a Mach output signal. The plugin that owns this register is notified with a **MSG_REG_CHANGED** synchronous message. All other plugins and the GUI are notified asynchronously with **MSG_REG_CHANGED**.

REG_TYPE_HOLDING specifies a general purpose register that cannot be mapped to either an input or output signal. The plugin that owns this register is notified with a **MSG_REG_CHANGED** synchronous message. All other plugins and the GUI are notified asynchronously with **MSG_REG_CHANGED**. This type of register usually contains boolean data.

REG_TYPE_COMMAND specifies a special purpose register that can be used to implement a simple command/response communication mechanism. Only the plugin that owns the register is notified with a **MSG_REG_COMMAND** synchronous message. It works with

- mcRegSendCommand

on the client side of the communication and

- mcRegGetCommand

and

- mcRegSetResponse

on the plguin side.

REG_TYPE_ENCODER specifies a register that can be mapped to a MPG or used to display the encoder counts in the GUI. Only Mach and the GUI are notified with a **MSG_REG_CHANGED** synchronous message.

Usage:

```
HMCREG m_hReg;
HMCDEV m_hDev; // Contains the device handle from the device registration.
int rc = mcRegRegister(m_hDev, "HoldingReg1", "HoldingReg1", REG_TYPE_HOLDING, &
```

mcRegSendCommand

C/C++ Syntax:

```
int mcRegSendCommand(
  HMCREG hReg,
  const char *command,
  char *response,
  size_t responseLen);
```

LUA Syntax:

```
response, rc = mc.mcRegSendCommand(
  hReg,
  command)
```

Description: Sends a text command to a plugin register.

Parameters:

Parameter	Description
hReg	A HMCREG specifying the register handle.
command	A string buffer specifying the command to be sent.
response	A string buffer that receives the response.
responseLen	The length of the response buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	**command** or **response** is NULL.
MERROR_INVALID_TYPE	The register is not of type **REG_TYPE_COMMAND**.
MERROR_REG_NOT_FOUND	The **hReg** parameter was 0 or invlaid.

Notes:
This function would primarily be used in scripts to control device features or provide a direct communication mechanism to the device.

The buffer pointed to by cmd cannot exceed 1024 bytes.

See Also:

- mcRegGetCommand

and

- mcRegSetResponse

Usage:

```
// Send a command via a register.
MINSTANCE mInst = 0;
if (mcRegGetHandle(mInst, "Sim0/SimCommand", &hReg;) == MERROR_NOERROR) {
 char response[1024];
 rc = mcRegSendCommand(hReg, "THC ON", response, sizeof(response));
 if (rc == MERROR_NOERROR) {
  // check response.
 }
}
```

mcRegSetDesc

C/C++ Syntax:

```
int mcRegSetDesc(
  HMCREG hReg,
  const char *desc);
```

LUA Syntax:

```
rc = mc.mcRegSetDesc(
  number hReg,
  string desc)
```

Description: Sets the description of the given ragister.

Parameters:

Parameter	Description
hReg	A HMCREG specifying the register handle.
desc	A string buffer specifying the register description.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_REG_NOT_FOUND	The **hReg** parameter was 0 or invlaid.
MERROR_INVALID_ARG	**desc** is NULL.

Notes:
None.

Usage:

```
// Set/change the description of the register.
MINSTANCE mInst = 0;
if (mcRegGetHandle(mInst, "core/global/Version", &hReg;) == MERROR_NOERROR) {
 rc = mcRegSetDesc(hReg, "Core Version");
}
```

mcRegSetName

C/C++ Syntax:

```
int mcRegSetName(
  HMCREG hReg,
  const char *name);
```

LUA Syntax:

```
rc = mc.mcRegSetName(
  number hReg,
  string name)
```

Description: Set the name of the given register.

Parameters:

Parameter	Description
hReg	A HMCREG specifying the register handle.
name	A string buffer specifying the name of the register.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_REG_NOT_FOUND	The **hReg** parameter was 0 or invlaid.
MERROR_INVALID_ARG	**name** is NULL.

Notes:
None.

Usage:

```
// Set/change the name of the register.
MINSTANCE mInst = 0;
if (mcRegGetHandle(mInst, "core/global/Version", &hReg;) == MERROR_NOERROR) {
 rc = mcRegSetName(hReg, "Version2");
 // The register path is now "core/global/Version2"
}
```

mcRegSetResponse

C/C++ Syntax:

```
int mcRegSetResponse(
  HMCREG hReg,
  char *response);
```

LUA Syntax:

```
N/A
```

Description: Set the reponse to the command register.

Parameters:

Parameter	Description
hReg	A HMCREG specifying the register handle.
response	A string buffer that specifies the response to the command.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_TYPE	The register is not of type **REG_TYPE_COMMAND**.
MERROR_REG_NOT_FOUND	The **hReg** parameter was 0 or invlaid.

Notes:

See Also:

- mcRegSendCommand

and

- mcRegGetCommand

Usage:

```
// Set the response to a register command.
int simControl::ProcessMsg(long msg, long param1, long param2)
{
```

```
 HMCREG hReg = (HMCREG)param1;
 long RegVal;
 switch (msg) {
 case MSG_REG_COMMAND:
  mcRegGetValueLong(hReg, &RegVal;);
  if (hReg = m_RegCommand) { // Is this our command register?
   char command[1024];
   mcRegGetCommand(hReg, command, sizeof(command));
   wxString cmd(command);
   cmd.MakeUpper();
   if (cmd == wxT("THC ON")) {
    m_thc = true;
    mcRegSetResponse(hReg, "OK");
   } else if (cmd == wxT("THC OFF")) {
    m_thc = false;
    mcMotionSync(m_cid);
    mcRegSetResponse(hReg, "OK");
   } else if (cmd == wxT("THC STATUS")) {
    if (m_thc) {
     mcRegSetResponse(hReg, "1");
    } else {
     mcRegSetResponse(hReg, "0");
    }
   }
  }
  break;
 default:
  ;
 }
 return(MERROR_NOERROR);
}
```

mcRegSetType

C/C++ Syntax:

```
int mcRegSetType(
  HMCREG hReg,
  int regType);
```

LUA Syntax:

```
rc = mc.mcRegSetType(
  number hReg,
  number regType)
```

Description: Set the type of the given register.

Parameters:

Parameter	Description
hReg	A HMCREG specifying the register handle.
regType	An integer specifying the register type. (REG_TYPE_NONE, IO_TYPE_REG_INPUT, REG_TYPE_OUTPUT, REG_TYPE_HOLDING, REG_TYPE_COMMAND, REG_TYPE_ENCODER)

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_REG_NOT_FOUND	The **hReg** parameter was 0 or invlaid.
MERROR_INVALID_ARG	**regType** is out of range.

Notes:
None.

Usage:

```
// Set the register type.
MINSTANCE mInst = 0;
if (mcRegGetHandle(mInst, "core/global/Version", &hReg;) == MERROR_NOERROR) {
 rc = mcRegSetType(hReg, REG_TYPE_HOLDING);
}
```

mcRegSetUserData

C/C++ Syntax:

```
int mcRegSetUserData(
  HMCREG hReg,
  void *data);
```

LUA Syntax:

```
N/A
```

Description: Set the user pointer for hReg.

Parameters:

Parameter	Description
hReg	A HMCREG specifying the register handle.
value	The address of a void pointer so user can have there own data.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_REG_NOT_FOUND	The **hReg** parameter was 0 or invlaid.
MERROR_INVALID_ARG	**data** is NULL.

Notes:
The hReg parameter **MUST** be a valid register handle. Otherwise, the function will crash.
This function allows the programmer to asscoiate private data that is local to the register's parent plugin/device.

Usage:

```
struct myDataStruct {
 int myInt;
 char myString[80];
 ...
};

HMCREG hReg;
MINSTANCE mInst = 0;
```

```
myDataStruct data;
// Get the handle to holding register 1 on SIM0.
mcRegGetHandle(mInst, "SIM0/HoldingReg1", &hReg;);
// Set the user data pointer to the address of data.
mcRegSetUserData(hReg, &data;);
```

mcRegSetValue

C/C++ Syntax:

```
int mcRegSetValue(
  HMCREG hReg,
  double value);
```

LUA Syntax:

```
rc = mc.mcRegSetValue(
  number hReg,
  number value)
```

Description: Sets the double value of the given register.

Parameters:

Parameter	Description
hReg	A HMCREG specifying the register handle.
value	A double specifying the register value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_REG_NOT_FOUND	The **hReg** parameter was 0 or invlaid.

Notes:
The hReg parameter **MUST** be a valid register handle. Otherwise, the function will crash.
This function should only be used upon a value change. It is a programming error otherwise and will cause messages to SPAM the core and GUI. MSG_REG_CHANGED is sent to the GUI and the plugin owning the device that owns the register.

Usage:

```
// Set the register value.
MINSTANCE mInst = 0;
if (mcRegGetHandle(mInst, "core/global/Version", &hReg;) == MERROR_NOERROR) {
 rc = mcRegSetValue(hReg, 4.0);
}
```

mcRegSetValueLong

C/C++ Syntax:

```
int mcRegSetValueLong(
  HMCREG hReg,
  double value);
```

LUA Syntax:

```
rc = mc.mcRegSetValueLong(
  number hReg,
  number value)
```

Description: Sets the long value of the given register.

Parameters:

Parameter	Description
hReg	A HMCREG specifying the register handle.
value	A long specifying the register value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_REG_NOT_FOUND	The **hReg** parameter was 0 or invlaid.

Notes:
The hReg parameter **MUST** be a valid register handle. Otherwise, the function will crash.
This function should only be used upon a value change. It is a programming error otherwise and will cause messages to SPAM the core and GUI. MSG_REG_CHANGED is sent to the GUI and the plugin owning the device that owns the register.

Usage:

```
// Set the register value.
MINSTANCE mInst = 0;
if (mcRegGetHandle(mInst, "core/global/Version", &hReg;) == MERROR_NOERROR) {
 rc = mcRegSetValueLong(hReg, 4);
}
```

mcRegSetValueString

C/C++ Syntax:

```
int mcRegSetValueString(
  HMCREG hReg,
  const char *value);
```

LUA Syntax:

```
rc = mc.mcRegSetValueString(
  number hReg,
  string value)
```

Description: Sets the string value of the given register.

Parameters:

Parameter	Description
hReg	A HMCREG specifying the register handle.
value	A string buffer specifying the register value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_REG_NOT_FOUND	The **hReg** parameter was 0 or invlaid.
MERROR_INVALID_ARG	**value** is NULL.

Notes:
The hReg parameter **MUST** be a valid register handle. Otherwise, the function will crash.
This function should only be used upon a value change. It is a programming error otherwise and will cause messages to SPAM the core and GUI. MSG_REG_CHANGED is sent to the GUI and the plugin owning the device that owns the register.

Usage:

```
// Set the register value.
MINSTANCE mInst = 0;
if (mcRegGetHandle(mInst, "core/global/Version", &hReg;) == MERROR_NOERROR) {
 rc = mcRegSetValueString(hReg, "4.0");
}
```

mcRegUnregister

C/C++ Syntax:

```
int mcRegUnregister(
  HMCDEV hDev,
  HMCREG hReg);
```

LUA Syntax:

```
N/A
```

Description: Remove a register from the core.

Parameters:

Parameter	Description
hDev	A HMCDEV specifying the register's parent device.
hReg	A HMCREG specifying the register to remove.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_DEVICE_NOT_FOUND	The **hDev** parameter was 0 or invlaid.
MERROR_REG_NOT_FOUND	The **hReg** parameter was 0 or invlaid.

Notes:
When register is removed all data in that register will be lost.

Usage:

```
MINSTANCE mInst = 0;
HMCREG hReg
char *path = "Sim0/Register1";

mcRegGetHandle(mInst, path, &hReg;);
mcRegUnregister(m_hDev, hReg);
```

mcScriptDebug

C/C++ Syntax:

```
int mcScriptDebug(
  MINSTANCE mInst,
  const char *filename);
```

LUA Syntax:

```
N/A
```

Description: Debug a script.

Parameters: (filename, A string buffer specifying the script file to debug.

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_PLUGIN_NOT_FOUND	There was no registered script plugin for the given script's extension.

Notes:
Script debugging is a function of script plugins. Script plugins register file extensions to identify what script can be used.

Usage:

```
// Debug the M6 macro
MINSTANCE mInst = 0;
int rc = mcScriptDebug(mInst, "m6.mcs");
```

mcScriptEngineRegister

C/C++ Syntax:

```
int mcScriptEngineRegister(
  MINSTANCE mInst,
  HMCPLUG plugid,
  const char *EngineName,
  const char *EngineDesc,
  const char *FileExtensions);
```

LUA Syntax:

```
N/A
```

Description:

Parameters:

Parameter	Description
mInst	The controller instance.
plugid	A HMCPLUG handle specifying the plugin.
EngineName	A string buffer specifying the script engine name.
EngineDesc	A string buffer specifying the script engine description.
FileExtensions	A string buffer specifying the file extenstions that the script engine uses. Each extension is separated with the pipe symbol "|".

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_PLUGIN_NOT_FOUND	**plugid** is invalid.

Notes:
Script plugins register file extensions to identify what script can be used.

Usage:

```
// Register the LUA script engine.
MINSTANCE mInst = 0;
HMCPLUG id; // From mcPluginInit();
int rc = mcScriptEngineRegister(mInst, id, "wxLua", "wxLua script engine", "mcs|
```

mcScriptExecute

C/C++ Syntax:

```
int mcScriptExecute(
  MINSTANCE mInst,
  const char *filename,
  BOOL async);
```

LUA Syntax:

```
rc = mc.mcScriptExecute(
  number mInst,
  string filename,
  number async)
```

Description: Execute a script in the global script context.

Parameters:

Parameter	Description
mInst	The controller instance.
filename	A string buffere specifying the script file to execute.
async	A BOOL specifying whether the script should be executed asynchronously (TRUE == no waiting on completion

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_PLUGIN_NOT_FOUND	No registered script plugins where found for the the specified script's file name extension.
MERROR_NOT_COMPILED	If the script engine compiles the scripts before run-time

Notes:
No error is returned if the specified script file does not exist. If the file does not exist, or there is a

sharing violation preventing the file from being read, this API function simply returns as a NO-OP.

Usage:

```
// Execute the M6 script and wait on it to complete.
MINSTANCE mInst = 0;
rc = mcScriptExecute(mInst, "m6.mcs", FALSE);
```

mcScriptExecuteIfExists

C/C++ Syntax:

```
int mcScriptExecuteIfExists(
  MINSTANCE mInst,
  const char *filename,
  BOOL async);
```

LUA Syntax:

```
rc = mc.mcScriptExecuteIfExists(
  number mInst,
  string filename,
  number async)
```

Description: Execute a script in the global script context if the specified script exists in the filesystem.

Parameters:

Parameter	Description
mInst	The controller instance.
filename	A string buffere specifying the script file to execute.
async	A BOOL specifying whether the script should be executed asynchronously (TRUE == no waiting on completion

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_PLUGIN_NOT_FOUND	No registered script plugins where found for the the specified script's file name extension.
MERROR_NOT_COMPILED	If the script engine compiles the scripts before run-time, this error may be returned which would be an indication of a script syntax error.

Notes:
No error is returned if the specified script file does not exist. If the file does not exist, or there is a sharing violation preventing the file from being read, this API function simply returns as a NO-OP.

Usage:

```
// Execute the M6 script and wait on it to complete.
MINSTANCE mInst = 0;
rc = mcScriptExecute(mInst, "m6.mcs", FALSE);
```

mcScriptExecutePrivate

C/C++ Syntax:

```
int mcScriptExecutePrivate(
  MINSTANCE mInst,
  const char *filename,
  BOOL async);
```

LUA Syntax:

```
rc = mc.mcScriptExecutePrivate(
  number mInst,
  string filename,
  BOOL async)
```

Description: Execute a script in the private script context.

Parameters:

Parameter	Description
mInst	The controller instance.
filename	A string buffere specifying the script file to execute.
async	A BOOL specifying whether the script should be executed asynchronously (TRUE == no waiting on completion

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_PLUGIN_NOT_FOUND	No registered script plugins where found for the the specified script's file name extension.
MERROR_NOT_COMPILED	If the script engine compiles the scripts before run-time, this error may be returned which would be an indication of a script syntax error.

Notes:
No error is returned if the specified script file does not exist. If the file does not exist, or there is a sharing violation preventing the file from being read, this API function simply returns as a NO-OP.

Usage:

```
// Execute myScript.mcs asynchronously in it's own private environment.
MINSTANCE mInst = 0;
int rc = mcScriptExecutePrivate(mInst, "myScript.mcs", TRUE);
```

mcScriptGetExtensions

C/C++ Syntax:

```
int mcScriptGetExtensions(MINSTANCE mInst,
  char *buf,
  long bufsize);
```

LUA Syntax:

```
N/A
```

Description: Retrieves all registered script engine file name extensions.

Parameters:

Parameter	Description
mInst	The controller instance.
buf	A string buffer to receive the script file name extensions.
bufsize	The length of the string buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**buf** is NULL or **bufsize** is <= 0.

Notes:
All returned extensions are delimited by pipe symbols ("|").

Usage:

```
// Retrieve all of the registered script extensions.
MINSTANCE mInst = 0;
mcScriptGetExtensions(MINSTANCE mInst, char *buf, long bufsize);
```

mcSignalEnable

C/C++ Syntax:

```
int mcSignalEnable(
 HMCSIG hSig,
 BOOL enabled);
```

LUA Syntax:

```
rc = mc.mcSignalEnable(
 number hSig,
 number enabled)
```

Description: Enable or disable a signal mapping.

Parameters:

Parameter	Description
hSig	A sginal handle obtained from mcSignalGetHandle().
enabled	TRUE to enable, FALSE to disable.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The signal parameter was 0.

Notes:
The hSig parameter **MUST** be a valid signal handle. Otherwise, the function will crash.
Signals have to be mapped to an IO handle before they can be enabled. It is important to note that the signal itself is always enabled. What is enabled it the signal mapping.

Usage:

```
simControl::simControl(MINSTANCE mInst, HMCPLUG id)
{
 m_cid = mInst;
 m_id = id;
 m_timer = new simTimer(this);
 m_cycletime = .001;
 HMCSIG hSig;

 mcDeviceRegister(m_cid, m_id, "Sim0", "Simulation Device", DEV_TYPE_MOTION | DE
```

```
 mcIoRegister(m_hDev, "Input0", "Input0", IO_TYPE_INPUT, &m;_Input0);
 mcIoRegister(m_hDev, "Output0", "Output0", IO_TYPE_OUTPUT, &m;_Output0)

 if (mcSignalGetHandle(m_cid, ISIG_INPUT1, int sigid, &hSig;) == MERROR_NOERROR)
  if (mcSignalMap(hSig, m_Input0) == MERROR_NOERROR) {
   // Signal mapped successfuly
mcSignalEnable(hSig, TRUE); // Enable the signal.
  }
 }
}
```

mcSignalGetHandle

C/C++ Syntax:

```
mcSignalGetHandle(
 MINSTANCE mInst,
 int sigid,
 HMCSIG *hSig);
```

LUA Syntax:

```
hSig, rc = mc.mcSignalGetHandle(
 number mInst,
 number sigid)
```

Description: Register the IO device in Mach4Core so it can be mapped by the user and other plugins

Parameters:

Parameter	Description
mInst	The controller instance.
sigid	The integer ID of the desired signal.
hSig	A pointer to a HMCSIG to receive the sginal handle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_SIGNAL_NOT_FOUND	The signal identified by sigid was not found.

Notes:
None.

Usage:

```
simControl::simControl(MINSTANCE mInst, HMCPLUG id)
{
 m_cid = mInst;
 m_id = id;
 m_timer = new simTimer(this);
```

```
 m_cycletime = .001;
 HMCSIG hSig;

 mcDeviceRegister(m_cid, m_id, "Sim0", "Simulation Device", DEV_TYPE_MOTION | DE

 mcIoRegister(m_hDev, "Input0", "Input0", IO_TYPE_INPUT, &m;_Input0);
 mcIoRegister(m_hDev, "Output0", "Output0", IO_TYPE_OUTPUT, &m;_Output0)

 if (mcSignalGetHandle(m_cid, ISIG_INPUT1, &hSig;) == MERROR_NOERROR) {
  if (mcSignalMap(hSig, m_Input0) == MERROR_NOERROR) {
   // Signal mapped successfuly.
   mcSignalEnable(hSig, true); // Enable the signal.
  }
 }
}
```

mcSignalGetInfo

C/C++ Syntax:

```
mcSignalGetInfo(
 HMCSIG hSig,
 int *enabled,
 char *name,
 size_t namelen,
 char *desc,
 size_t desclen,
 int *activelow);
```

LUA Syntax:

```
enabled, name, desc, activelow, rc = mc.mcSignalGetInfo(number hSig)
```

Description: Return infromation about a signal.

Parameters:

Parameter	Description
hSig	The handle for the signal.
enabled	The address of an integer to receive the enabled status of the signal.
name	A buffer to receive the name of the signal.
namelen	A value specifiying the length of the name buffer.
desc	A buffer to receive the description of the signal.
desclen	A value specifiying the length of the description buffer.
activelow	The address of an integer to receive the active low status of the signal.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The hSig parameter was 0.

Notes:
The hSig parameter **MUST** be a valid signal handle. Otherwise, the function will crash.

Usage:

```
HMCSIG hSig = 0;
int enabled = 0;
int nameLen = 20
char nameBuf[nameLen];
int descLen = 40;
char descBuf[descLen];
int activeLow = 0;

while (mcSignalGetNextHandle(0, SIG_TYPE_INPUT, hSig, &hSig;) == MERROR_NOERROR)
 if (hSig != 0) {
  mcSignalGetInfo(hSig, &enabled;, nameBuf, nameLen, descBuf, descLen, &activeLo
 } else {
  break;
 }
}
```

mcSignalGetInfoStruct

C/C++ Syntax:

```
mcSignalGetInfoStruct(
 HMCSIG hSig,
 siginfo_t *siginf);
```

LUA Syntax:

```
N/A
```

Description: Return infromation about a signal.

Parameters:

Parameter	Description
hSig	The handle for the signal.
siginf	The adress of a siginfo struct to receive the signal information.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
No Error.	
MERROR_INVALID_ARG	The hSig parameter was 0.

Notes:
The hSig parameter **MUST** be a valid signal handle. Otherwise, the function will crash.

```
struct siginfo {
 char sigName[80];
 char sigDesc[80];
 int  sigEnabled;
 int  sigActiveLow;
 HMCIO sigMappedIoHandle;
};
typedef struct siginfo siginfo_t;
```

Usage:

```
HMCSIG hSig = 0;
```

```
siginfo_t siginf;

while (mcSignalGetNextHandle(0, SIG_TYPE_INPUT, hSig, &hSig;) == MERROR_NOERROR)
 if (hSig != 0) {
  mcSignalGetInfoStruct(hSig, &siginf;);
 } else {
  break;
 }
}
```

mcSignalGetNextHandle

C/C++ Syntax:

```
mcSignalGetNextHandle(
 MINSTANCE mInst,
 int sigtype,
 HMCSIG startSig,
 HMCSIG *hSig);
```

LUA Syntax:

```
hSig, rc = mc.mcSignalGetNextHandle(
 number mInst,
 number sigtype,
 number startSig)
```

Description: Provides a means of looping through the available signals.

Parameters:

Parameter	Description
mInst	The controller instance.
sigtype	The signal type for which to look (SIG_TYPE_INPUT or SIG_TYPE_OUTPUT).
startSig	The current signal handle.
hSig	A pointer to a HMCSIG to receive the signal handle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	sigtype was not SIG_TYPE_INPUT or SIG_TYPE_OUTPUT.
MERROR_NODATA	No more signals can be found.

Notes:
To start, call mcSignalGetNextHandle() with startSig = 0.

Usage:

```
HMCSIG hSig = 0;

while (mcSignalGetNextHandle(0, SIG_TYPE_INPUT, hSig, &hSig;) == MERROR_NOERROR)
    if (hSig != 0) {
        // Do something with hSig.
    } else {
        break;
    }
}
```

mcSignalGetState

C/C++ Syntax:

```
mcSignalGetState(
 HMCSIG hSig,
 BOOL *state)
```

LUA Syntax:

```
state, rc = mc.mcSignalGetState(
 number hSig)
```

Description: Set the state of a signal.

Parameters:

Parameter	Description
hSig	The handle for the signal.
state	Defines the desired state of the signal. TRUE is active

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The hSig parameter was 0 or invalid.

Notes:
The hSig parameter **MUST** be a valid signal handle. Otherwise, the function will crash.

Usage:

```
HMCSIG hSig = 0;
BOOL sigState = FALSE;

// Get the state of all input signals.
while (mcSignalGetNextHandle(0, SIG_TYPE_INPUT, hSig, &hSig;) == MERROR_NOERROR)
    if (hSig != 0) {
        mcSignalGetState(hSig, &sigState;);
    } else {
        break;
    }
}
```

mcSignalMap

C/C++ Syntax:

```
mcSignalMap(
 HMCSIG hSig,
 HMCIO hIo);
```

LUA Syntax:

```
rc = mc.mcSignalMap(
 number hSig,
 number hIo)
```

Description: Map a core signal to a registered I/O handle.

Parameters:

Parameter	Description
hSig	A sginal handle obtained from mcSignalGetHandle().
hIo	A registered IO handle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The signal parameter was 0.

Notes:
The hSig and hIo parameters **MUST** be a valid handles. Otherwise, the function will crash.
Mapping an input IO handle to an output signal handle or an output IO handle to an input signal handle is a programming error.

Usage:

```
simControl::simControl(MINSTANCE mInst, HMCPLUG id)
{
 m_cid = mInst;
 m_id = id;
 m_timer = new simTimer(this);
 m_cycletime = .001;
 HMCSIG hSig;

 mcDeviceRegister(m_cid, m_id, "Sim0", "Simulation Device", DEV_TYPE_MOTION | DE
```

```
 mcIoRegister(m_hDev, "Input0", "Input0", IO_TYPE_INPUT, &m;_Input0);
 mcIoRegister(m_hDev, "Output0", "Output0", IO_TYPE_OUTPUT, &m;_Output0)

 if (mcSignalGetHandle(m_cid, ISIG_INPUT1, &hSig;) == MERROR_NOERROR) {
  if (mcSignalMap(hSig, m_Input0) == MERROR_NOERROR) {
   // Signal mapped successfuly
  }
 }
}
```

mcSignalSetActiveLow

C/C++ Syntax:

```
mcSignalSetActiveLow(
 HMCSIG hSig,
 BOOL activelow);
```

LUA Syntax:

```
rc = mc.mcSignalSetActiveLow(
 number hSig,
 number activelow)
```

Description: Set the state of a signal.

Parameters:

Parameter	Description
hSig	The handle for the signal.
activelow	TRUE to set the signal as negated.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The hSig parameter was 0 or invalid.

Notes:
The hSig parameter **MUST** be a valid signal handle. Otherwise, the function will crash.

Usage:

```
HMCSIG hSig = 0;
BOOL activeLow = TRUE;

// Set all input signals active low.
while (mcSignalGetNextHandle(0, SIG_TYPE_INPUT, hSig, &hSig;) == MERROR_NOERROR)
 if (hSig != 0) {
  mcSignalSetActiveLow(hSig, activeLow);
 } else {
  break;
 }
}
```

mcSignalSetState

C/C++ Syntax:

```
mcSignalSetState(
 HMCSIG hSig,
 BOOL enabled);
```

LUA Syntax:

```
rc = mc.mcSignalSetState(
 number hSig,
 number enabled)
```

mcSignalSetState(HMCSIG hSig, bool enabled); **Description:**
Set the state of a signal.

Parameters:

Parameter	Description
hSig	The handle for the signal.
enabled	TRUE is active

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The hSig parameter was 0.

Notes:
The hSig parameter **MUST** be a valid signal handle. Otherwise, the function will crash.

Usage:

```
HMCSIG hSig = 0;

// Set all output signals inactive.
while (mcSignalGetNextHandle(0, SIG_TYPE_OUTPUT, hSig, &hSig;) == MERROR_NOERROR)
 if (hSig != 0) {
  mcSignalSetState(hSig, FALSE);
 } else {
  break;
 }
}
```

mcSignalUnmap

C/C++ Syntax:

```
mcSignalUnmap(
 HMCSIG hSig);
```

LUA Syntax:

```
rc = mc.mcSignalUnmap(
 number hSig)
```

Description: Unmap any I/O objects from the specified core signal.

Parameters:

Parameter	Description
hSig	A sginal handle obtained from mcSignalGetHandle().

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_ARG	The signal parameter was 0 or invalid.

Notes:
The hSig parameters **MUST** be a valid handle. Otherwise, the function will crash.

Usage:

```
MINSTANCE mInst = 0;
HMCSIG hSig;

if (mcSignalGetHandle(mInst, ISIG_INPUT1, &hSig;) == MERROR_NOERROR) {
 if (mcSignalUnmap(hSig) == MERROR_NOERROR) {
  // Signal now has no I/O mappings
 }
}
```

mcSignalWait

C/C++ Syntax:

```
mcSignalWait(
 MINSTANCE mInst,
 int sigId,
 int waitMode,
 double timeoutSecs);
```

LUA Syntax:

```
rc = mc.mcSignalWait(
 number mInst,
 number sigId,
 number waitMode,
 number timeoutSecs);
```

Description: Wait on a signal to change state.

Parameters:

Parameter	Description
hSig	A sginal handle obtained from mcSignalGetHandle().
sigId	A valid signal ID.
waitMode	An integer specifying whether to wait on the signal to go high (WAIT_MODE_HIGH) or low (WAIT_MODE_LOW).
timeoutSecs	A double specifying a timeout period.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**sigId** or **wiatMode** is out of range or **timeoutSecs** is negative.
MERROR_NOT_ENABLED	The control is not enabled.
MERROR_TIMED_OUT	The timeout period was reached without a change of state.

Notes:

Specifying 0 for **timeoutSecs** will wait indefinitely until the signal changes state. Decimals are allowed. e.g. .5 is half of a second.)

If the control is diabled while waiting on a signal to change state, the function stops waiting and returns MERROR_NOT_ENABLED.

Usage:

```
MINSTANCE mInst = 0;
HMCSIG hSig;
int rc;

rc = mcSignalWait(mInst, ISIG_INPUT1, WAIT_MODE_HIGH, .5);
switch (rc) {
case MERROR_NOERROR:
 // Signal changed state from low to high
 break;
case MERROR_TIMED_OUT:
 // The signal didn't change state in the alotted time.
 break;
case MERROR_NOT_ENABLED:
 // The control was not enabled at the time of the
   function call or the control was disabled during the function call.
 break;
}
```

mcSoftLimitGetState

C/C++ Syntax:

```
mcSoftLimitGetState(
 MINSTANCE mInst,
 int axis,
 double *ison);
```

Description: Used to check if the Softlimit for an axis is on or off.

Parameters:

Parameter	Description
mInst	The controller instance.
axis	The axis from which to to get the Softlimit state.
ison	A pointer to a double to receive the status of Softlimit (on or off).

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Can be used to display to the user what the status of the Softlimit for an axis is.

Usage:

```
int mInst = 0;
int axis = Z_AXIS;
dobule IsOn = 0; // DO_OFF to turn off.
mcSoftLimitSetState(mInst, axis, &IsOn;); // Get the softlimit for the Z axis ret
```

mcSoftLimitMaxMinsClear

C/C++ Syntax:

```
mcSoftLimitMaxMinsClear(
 MINSTANCE mInst);
```

Description: Clear the Softlimits for a loaded Gcode file

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Clear the Min and Max SoftLimits for a File. This should be done by modules that do the toolpath display.

Usage:

```
int mInst=0;
// Clear the Softlimits for the file so they will be recalculated when a regen is
mcSoftLimitMaxMinsClear(mInst);
```

mcSoftLimitSetState

C/C++ Syntax:

```
mcSoftLimitSetState(
 MINSTANCE mInst,
 int axis,
 int on);
```

Description: Set softlimits to be on or off for an axis.

Parameters:

Parameter	Description
mInst	The controller instance.
axis	The axis to set the softlimit to be on or off.
on	Set the softlimit to be on or off (MC_ON, MC_OFF).

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_AXIS_NOT_FOUND	The axis specified by **axisId** was not found.

Notes:
Can be used at any time. If you enable the softlimits when you are out of bounds any jog will move it back to the softlimit.

Usage:

```
int mInst=0;
int axis = Z_AXIS;
int TurnOn = MC_ON; // MC_OFF to turn off.
mcSoftLimitSetState(mInst, axis, TurnOn); // Set the softlimit forthe Z axis to
```

mcSpindleCalcCSSToRPM

C/C++ Syntax:

```
int mcSpindleCalcCSSToRPM(
  MINSTANCE mInst,
  double DiaOfCut,
  BOOL Inch);
```

LUA Syntax:

```
rc = mc.mcSpindleCalcCSSToRPM(
  number mInst,
  number DiaOfCut,
  number Inch);
```

Description: A utility function to implement Constant Surface Speed (CSS) for lathe operation.

Parameters:

Parameter	Description
mInst	The controller instance.
DiaOfCut	A double specifying the diameter of the next cut.
inch	A BOOL specifying if the diameter is in inches (TRUE) or millimiters (FALSE).

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_NOW	The spindle is not in CSS mode.

Notes:
None.

Usage:

```
// make the spindle speed calc the correct RPM for a .550" inch cut.
MINSTANCE mInst = 0;
int rc = mcSpindleCalcCSSToRPM(mInst, .550, TRUE);
```

mcSpindleGetAccelTime

C/C++ Syntax:

```
int mcSpindleGetAccelTime(
  MINSTANCE mInst,
  int Range,
  double *Sec);
```

LUA Syntax:

```
Sec, rc = mc.mcSpindleGetAccelTime(
  number mInst,
  number Range)
```

Description: Retrieve the spindle acceleration time in seconds for the given spindle range.

Parameters:

Parameter	Description
mInst	The controller instance.
Range	An integer specifying the spindle range.
Sec	The address to a double to receive the acceleration time.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_SPIN_RANGE_NOT_FOUND	**Range** specified an invalid spindle range.
MERROR_INVALID_ARG	**Sec** cannot be NULL.

Notes:
None.

Usage:

```
// Get the accel time for spindle range 2.
MINSTANCE mInst = 0;
```

```
double sec = 0;
int rc = mcSpindleGetAccelTime(mInst, 2, &sec;);
```

mcSpindleGetCommandRPM

C/C++ Syntax:

```
int mcSpindleGetCommandRPM(
  MINSTANCE mInst,
  double *RPM);
```

LUA Syntax:

```
RPM, rc = mc.mcSpindleGetCommandRPM(
  number mInst)
```

Description: Retrieve the commanded RPM.

Parameters:

Parameter	Description
mInst	The controller instance.
RPM	The address of a double to receive the commanded RPM.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**RPM** cannot be NULL.

Notes:
None.

Usage:

```
// Get the current commanded RPM
MINSTANCE mInst = 0;
double rpm = 0.0;
int rc = mcSpindleGetCommandRPM(mInst, &rpm;);
```

mcSpindleGetCurrentRange

C/C++ Syntax:

```
int mcSpindleGetCurrentRange(
  MINSTANCE mInst,
  int *Range);
```

LUA Syntax:

```
Range, rc = mc.mcSpindleGetCurrentRange(
  number mInst)
```

Description: Retrieve the current spindle range.

Parameters:

Parameter	Description
mInst	The controller instance.
Range	The address of an integer to receive the current spindle range.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**Range** cannot be NULL.

Notes:
None.

Usage:

```
// Get the current spindle range.
MINSTANCE mInst = 0;
int range = 0;
int rc = mcSpindleGetCurrentRange(mInst, □);
```

mcSpindleGetDecelTime

C/C++ Syntax:

```
int mcSpindleGetDecelTime(
  MINSTANCE mInst,
  int Range,
  double *Sec);
```

LUA Syntax:

```
Sec, rc = mc.mcSpindleGetDecelTime(
  number mInst,
  number Range)
```

Description: Retrieve the decel. time in seconds for the given spindle range.

Parameters:

Parameter	Description
mInst	The controller instance.
Range	An integer specifying the spindle range.
Sec	The address of a double to receive the decel. time.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_SPIN_RANGE_NOT_FOUND	**Range** specified an invalid spindle range.
MERROR_INVALID_ARG	**Sec** cannot be NULL.

Notes:
None.

Usage:

```
// Get the decel. time for spindle range 3
MINSTANCE mInst = 0;
double sec = 0.0;
```

```
int rc = mcSpindleGetDecelTime(mInst, 3, &ec;);
```

mcSpindleGetDirection

C/C++ Syntax:

```
int mcSpindleGetDirection(
  MINSTANCE mInst,
  int *dir);
```

LUA Syntax:

```
dir, rc = mc.mcSpindleGetDirection(
  number mInst)
```

Description: Retrieve the current spindle direction.

Parameters:

Parameter	Description
mInst	The controller instance.
dir	An address to an integer to receive the current spindle direction. (MC_SPINDLE_OFF, MC_SPINDLE_FWD, and MC_SPINDLE_REV)

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**dir** cannot be NULL.

Notes:
None.

Usage:

```
// Get the current spindle direction.
MINSTANCE mInst = 0;
int dir = MC_SPINDLE_OFF;
int rc = mcSpindleGetDirection(mInst, &dir;);
```

mcSpindleGetFeedbackRatio

C/C++ Syntax:

```
int mcSpindleGetFeedbackRatio(
  MINSTANCE mInst,
  int Range,
  double *Ratio);
```

LUA Syntax:

```
Ratio, rc = mc.mcSpindleGetFeedbackRatio(
  number mInst,
  number Range)
```

Description: Retrieve the spindle feedback ratio for the given spindle range.

Parameters:

Parameter	Description
mInst	The controller instance.
Range	An integer specifying the spindle range.
Ratio	The address of a double to receive the feedback ratio.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_SPIN_RANGE_NOT_FOUND	**Range** specified an invalid spindle range.
MERROR_INVALID_ARG	**Ratio** cannot be NULL.

Notes:
None.

Usage:

```
// Get the feedback ratio for spindle range 0.
MINSTANCE mInst = 0;
double ratio = 0.0;
```

```
int rc = mcSpindleGetFeedbackRatio(mInst, 0, :);
```

mcSpindleGetMaxRPM

C/C++ Syntax:

```
int mcSpindleGetMaxRPM(
  MINSTANCE mInst,
  int Range,
  double *MaxRPM);
```

LUA Syntax:

```
MaxRPM, rc = mc.mcSpindleGetMaxRPM(
  number mInst,
  number Range)
```

Description: Retrieve the maximum RPM for the given spindle range.

Parameters:

Parameter	Description
mInst	The controller instance.
Range	An integer specifying the spindle range.
MaxRPM	the address of a double to receive the maximum RPM for the given spindle range.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_SPIN_RANGE_NOT_FOUND	**Range** specified an invalid spindle range.
MERROR_INVALID_ARG	**MaxRPM** cannot be NULL.

Notes:
None.

Usage:

```
// Get the max RPM for range 0.
MINSTANCE mInst = 0;
```

```
double mrpm = 0.0
int rc = mcSpindleGetMaxRPM(mInst, 0, &mrpm;);
```

mcSpindleGetMinRPM

C/C++ Syntax:

```
int mcSpindleGetMinRPM(
  MINSTANCE mInst,
  int Range,
  double *MinRPM);
```

LUA Syntax:

```
MinRPM, rc = mc.mcSpindleGetMinRPM(
  number mInst,
  number Range)
```

Description: Retrieve the minimum RPM for the given spindle range.

Parameters:

Parameter	Description
mInst	The controller instance.
Range	An integer specifying the spindle range.
MaxRPM	the address of a double to receive the minimum RPM for the given spindle range.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_SPIN_RANGE_NOT_FOUND	**Range** specified an invalid spindle range.
MERROR_INVALID_ARG	**MinRPM** cannot be NULL.

Notes:
None.

Usage:

```
// Get the min RPM for range 0.
MINSTANCE mInst = 0;
```

```
double mrpm = 0.0
int rc = mcSpindleGetMinRPM(mInst, 0, &mrpm;);
```

mcSpindleGetMotorAccel

C/C++ Syntax:

```
int mcSpindleGetMotorAccel(
  MINSTANCE mInst,
  int Range,
  double *Accel);
```

LUA Syntax:

```
Accel, rc = mc.mcSpindleGetMotorAccel(
  number mInst,
  number Range)
```

Description: Retrieve the spindle motor acceleration value.

Parameters:

Parameter	Description
mInst	The controller instance.
Range	An integer specifying the spindle range.
Accel	The address to a deouble to receive the spindle acceleration for the given range.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_SPIN_RANGE_NOT_FOUND	**Range** specified an invalid spindle range.
MERROR_INVALID_ARG	**Accel** cannot be NULL.

Notes:
None.

Usage:

```
// Get the spindle acceleration for range 1.
MINSTANCE mInst = 0;
```

```
double accel = 0.0;
int rc = mcSpindleGetMotorAccel(mInst, 1, &accel;);
```

mcSpindleGetMotorMaxRPM

C/C++ Syntax:

```
int mcSpindleGetMotorMaxRPM(
  MINSTANCE mInst,
  double *RPM);
```

LUA Syntax:

```
RPM, rc = mc.mcSpindleGetMotorMaxRPM(
  number mInst)
```

Description: Retrieve the maximum defined RPM for the spindle motor.

Parameters:

Parameter	Description
mInst	The controller instance.
RPM	The address of a double to receive the RPM.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**RPM** cannot be NULL.

Notes:
None.

Usage:

```
// Get the maximum spindle motor RPM.
MINSTANCE mInst = 0;
double rpm = 0.0;
int rc = mcSpindleGetMotorMaxRPM(mInst, &rpm;);
```

mcSpindleGetMotorRPM

C/C++ Syntax:

```
int mcSpindleGetMotorRPM(
  MINSTANCE mInst,
  double *RPM);
```

LUA Syntax:

```
rpm, rc = mc.mcSpindleGetMotorRPM(
  number mInst)
```

Description: Retrieve the current motor RPM.

Parameters:

Parameter	Description
mInst	The controller instance.
RPM	The address to a double to receive the motor RPM.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**RPM** cannot be NULL.

Notes:
This is not the spidnle RPM!!! It is the motor RPM without any ratios for ranges (pulleys), etc...

Usage:

```
// Get the current motor RPM.
MINSTANCE mInst = 0;
double rpm = 0.0;
int rc = mcSpindleGetMotorRPM(mInst, &rpm;);
```

mcSpindleGetOverride

C/C++ Syntax:

```
int mcSpindleGetOverride(
  MINSTANCE mInst,
  double *percent);
```

LUA Syntax:

```
percent, rc = mcSpindleGetOverride(
  number mInst)
```

Description: Retrieve the current spindle override value as a percentage of the commanded RPM.

Parameters: ;

Parameter	Description
mInst	The controller instance.
percent	The address to a double to receive the override decimal percentage.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Get the spindle override. .5 == 50%
MINSTANCE mInst = 0;
double percent = 0.0;
int rc = mcSpindleGetOverride(mInst, &percent;);
```

mcSpindleGetReverse

C/C++ Syntax:

```
int mcSpindleGetReverse(
  MINSTANCE mInst,
  int Range,
  BOOL *Reversed);
```

LUA Syntax:

```
Rreversed, rc = mc.mcSpindleGetReverse(
  number mInst,
  number Range)
```

Description: Retrieve whether the spindle is reversed for the given spindle range.

Parameters:

Parameter	Description
mInst	The controller instance.
Range	An integer specifying the spindle range.
Reversed	The address of a BOOL to receive if the spindle is reversed for the given range.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_SPIN_RANGE_NOT_FOUND	**Range** specified an invalid spindle range.
MERROR_INVALID_ARG	**Reversed** cannot be NULL.

Notes:
None.

Usage:

```
// See if the spindle is reversed in range 1
MINSTANCE mInst = 0;
```

```
BOOL reversed = FALSE;
int mcSpindleGetReverse(mInst, 1, &reversed;);
```

mcSpindleGetSensorRPM

C/C++ Syntax:

```
int mcSpindleGetSensorRPM(
  MINSTANCE mInst,
  double *RPM);
```

LUA Syntax:

```
RPM, rc = mc.mcSpindleGetSensorRPM(
  number mInst)
```

Description: Retrieve the RPM at the spindle sensor.

Parameters:

Parameter	Description
mInst	The controller instance.
RPM	The address of a double to receive the RPM.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**RPM** cannot be NULL.

Notes:
This API function will report RPM of the spindle at the spindle sensor and does not include any feedback ratio.

Usage:

```
// Get the spindle sensor RPM.
MINSTANCE mInst = 0;
double rpm = 0.0;
int rc = mcSpindleGetSensorRPM(mInst, &rpm;);
```

mcSpindleGetSpeedCheckEnable

C/C++ Syntax:

```
int mcSpindleGetSpeedCheckEnable(
  MINSTANCE mInst,
  BOOL *enabled);
```

LUA Syntax:

```
enabled, rc = mc.mcSpindleGetSpeedCheckEnable(
  number mInst)
```

Description: Determine if spindle speed check is enabled.

Parameters:

Parameter	Description
mInst	The controller instance.
enabled	The address of a BOOL that receives the spindle speed check status.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**enabled** cannot be NULL.

Notes:
None.

Usage:

```
// Check the staus of spindle speed check.
MINSTANCE mInst = 0;
BOOL enabled = FALSE;
int rc = mcSpindleGetSpeedCheckEnable(mInst, &enabled;);
```

mcSpindleGetSpeedCheckPercent

C/C++ Syntax:

```
int mcSpindleGetSpeedCheckPercent(
  MINSTANCE mInst,
  double *percent);
```

LUA Syntax:

```
percent, rc = mc.mcSpindleGetSpeedCheckPercent(
  number mInst)
```

Description: Retrieve the spindle speed check percentage.

Parameters:

Parameter	Description
mInst	The controller instance.
percent	The address of a double to receive the speed check percentage.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**percent** cannot be NULL.

Notes:
None.

Usage:

```
// Get the speed check percentage.
MINSTANCE mInst = 0;
double percent = 0.0;
int rc = mcSpindleGetSpeedCheckPercent(mInst, &percent;);
```

mcSpindleGetTrueRPM

C/C++ Syntax:

```
int mcSpindleGetTrueRPM(
  MINSTANCE mInst,
  double *RPM);
```

LUA Syntax:

```
RPM, rc = mc.mcSpindleGetTrueRPM(
  number mInst)
```

Description: Retrieve the true RPM of the spindle, includeing any feedback ratio.

Parameters:

Parameter	Description
mInst	The controller instance.
RPM	The address of a double to receive the true RPM of the spindle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**RPM** cannot be NULL.

Notes:
Calling this API function includes any feedback ratio.

Usage:

```
// Get the true spindle RPM.
MINSTANCE mInst = 0;
double rpm = 0.0;
int rc = mcSpindleGetTrueRPM(mInst, &rpm;);
```

mcSpindleSetAccelTime

C/C++ Syntax:

```
int mcSpindleSetAccelTime(
  MINSTANCE mInst,
  int Range,
  double Sec);
```

LUA Syntax:

```
rc = mc.mcSpindleSetAccelTime(
  number mInst,
  number Range,
  number Sec);
```

Description: Sets the spindle accel. time for the given spindle range.

Parameters:

Parameter	Description
mInst	The controller instance.
Range	An integer specifying the spindle range.
Sec	A double specifying the time in seconds.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_SPIN_RANGE_NOT_FOUND	**Range** specified an invalid spindle range.

Notes:
None.

Usage:

```
// Set the accel. time for spindle range 0.
MINSTANCE mInst = 0;
int rc = mcSpindleSetAccelTime(mInst, 0, 5.0);
```

mcSpindleSetCommandRPM

C/C++ Syntax:

```
int mcSpindleSetCommandRPM(
  MINSTANCE mInst,
  double RPM);
```

LUA Syntax:

```
rc = mc.mcSpindleSetCommandRPM(
  number mInst,
  number RPM)
```

Description: Set the commanded RPM for the spindle.

Parameters:

Parameter	Description
mInst	The controller instance.
RPM	A double specifying the RPM.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Set the spindle RPM to 5000.
MINSTANCE mInst = 0;
int rc = mcSpindleSetCommandRPM(mInst, 5000);
```

mcSpindleSetDecelTime

C/C++ Syntax:

```
int mcSpindleSetDecelTime(
  MINSTANCE mInst,
  int Range,
  double Sec);
```

LUA Syntax:

```
rc = mc.mcSpindleSetDecelTime(
  number mInst,
  number Range,
  number Sec)
```

Description: Set the decel. time for the given spindle range.

Parameters:

Parameter	Description
mInst	The controller instance.
Range	An integer specifying the spindle range.
Sec	A double specifying the time in seconds.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_SPIN_RANGE_NOT_FOUND	**Range** specified an invalid spindle range.

Notes:
None.

Usage:

```
// Set the decel. time for spindle range 0.
MINSTANCE mInst = 0;
in rc = mcSpindleSetDecelTime(mInst, 0, 5);
```

mcSpindleSetDirection

C/C++ Syntax:

```
int mcSpindleSetDirection(
  MINSTANCE mInst,
  int dir);
```

LUA Syntax:

```
rc = mc.mcSpindleSetDirection(
  number mInst,
  number dir)
```

Description: Set the desired spindle direction.

Parameters:

Parameter	Description
mInst	The controller instance.
dir	An integer specifying the spindle direction. (MC_SPINDLE_OFF, MC_SPINDLE_FWD, and MC_SPINDLE_REV)

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_NOW	An attempt was made to set the direction to MC_SPINDLE_FWD or MC_SPINDLE_REV when the control is disabled.
MERROR_INVALID_ARG	**dir** is out of range.

Notes:
None.

Usage:

```
// Set the spindle direction to FWD
MINSTANCE mInst = 0;
```

```
int rc = mcSpindleSetDirection(mInst, MC_SPINDLE_FWD);
```

mcSpindleSetDirectionWait

C/C++ Syntax:

```
int mcSpindleSetDirectionWait(
  MINSTANCE mInst,
  int dir);
```

LUA Syntax:

```
rc = mc.mcSpindleSetDirectionWait(
  number mInst,
  number dir)
```

Description: Set the desired spindle direction and wait for the spindle to come to speed.

Parameters:

Parameter	Description
mInst	The controller instance.
dir	An integer specifying the spindle direction. (MC_SPINDLE_OFF, MC_SPINDLE_FWD, and MC_SPINDLE_REV)

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_NOW	An attempt was made to set the direction to MC_SPINDLE_FWD or MC_SPINDLE_REV when the control is disabled.
MERROR_INVALID_ARG	**dir** is out of range.

Notes:
None.

Usage:

```
// Set the spindle direction to FWD and wait
MINSTANCE mInst = 0;
```

```
int rc = mcSpindleSetDirectionWait(mInst, MC_SPINDLE_FWD);
```

mcSpindleSetFeedbackRatio

C/C++ Syntax:

```
int mcSpindleSetFeedbackRatio(
  MINSTANCE mInst,
  int Range,
  double Ratio);
```

LUA Syntax:

```
rc = mc.mcSpindleSetFeedbackRatio(
  number mInst,
  number Range,
  number Ratio)
```

Description: Set the feedback ratio for the given spindle range.

Parameters:

Parameter	Description
mInst	The controller instance.
Range	An integer specifying the spindle range.
Ratio	A double specifying the feedback ratio.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_SPIN_RANGE_NOT_FOUND	**Range** specified an invalid spindle range.

Notes:
None.

Usage:

```
// Set the feedback ratio for spindle range 0.
MINSTANCE mInst = 0;
int mcSpindleSetFeedbackRatio(mInst, 0, 1.2); // 1.2 : 1
```

mcSpindleSetMaxRPM

C/C++ Syntax:

```
int mcSpindleSetMaxRPM(
  MINSTANCE mInst,
  int Range,
  double RPM);
```

LUA Syntax:

```
rc = mc.mcSpindleSetMaxRPM(
  number mInst,
  number Range,
  number RPM);
```

Description: Set the max RPM for the given spindle range.

Parameters:

Parameter	Description
mInst	The controller instance.
Range	An integer specifying the spindle range.
RPM	A double specifying the maximum RPM.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_SPIN_RANGE_NOT_FOUND	**Range** specified an invalid spindle range.

Notes:
None.

Usage:

```
// Set the max RPM for spindle range 0 to 5000
MINSTANCE mInst = 0;
int mcSpindleSetMaxRPM(mInst, 0, 5000);
```

mcSpindleSetMinRPM

C/C++ Syntax:

```
int mcSpindleSetMinRPM(
  MINSTANCE mInst,
  int Range,
  double RPM);
```

LUA Syntax:

```
rc = mc.mcSpindleSetMinRPM(
  number mInst,
  number Range,
  number RPM);
```

Description: Set the min RPM for the given spindle range.

Parameters:

Parameter	Description
mInst	The controller instance.
Range	An integer specifying the spindle range.
RPM	A double specifying the minimum RPM.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_SPIN_RANGE_NOT_FOUND	**Range** specified an invalid spindle range.

Notes:
None.

Usage:

```
// Set the min RPM for spindle range 0 to 60
MINSTANCE mInst = 0;
int mcSpindleSetMinRPM(mInst, 0, 60);
```

mcSpindleSetMotorAccel

C/C++ Syntax:

```
int mcSpindleSetMotorAccel(
  MINSTANCE mInst,
  int Range,
  double Accel);
```

LUA Syntax:

```
rc = mc.mcSpindleSetMotorAccel(
  MINSTANCE mInst,
  int Range,
  double Accel);
```

Description: Set the spindle motor acceleration for the given spindle range.

Parameters:

Parameter	Description
mInst	The controller instance.
Range	An integer specifying the spindle range.
Accel	A double sepcifying the acceleration value.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_SPIN_RANGE_NOT_FOUND	**Range** specified an invalid spindle range.

Notes:
None.

Usage:

```
// Set the acceleration for the spindle motor for range 0
MINSTANCE mInst = 0;
int rc = mcSpindleSetMotorAccel(mInst, 0, 10000);
```

mcSpindleSetMotorMaxRPM

C/C++ Syntax:

```
int mcSpindleSetMotorMaxRPM(
  MINSTANCE mInst,
  double RPM);
```

LUA Syntax:

```
rc = mc.mcSpindleSetMotorMaxRPM(
  number mInst,
  number RPM)
```

Description: Set the maximum spindle motor RPM.

Parameters:

Parameter	Description
mInst	The controller instance.
RPM	A double specifying the maximum spindle motor RPM.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Cap the spindle motor RPM to 6000.
MINSTANCE mInst = 0;
int rc = mcSpindleSetMotorMaxRPM(mInst, 6000);
```

mcSpindleSetOverride

C/C++ Syntax:

```
int mcSpindleSetOverride(
  MINSTANCE mInst,
  double percent);
```

LUA Syntax:

```
rc = mc.mcSpindleSetOverride(
  number mInst,
  number percent)
```

Description: Set the spindle override percentage.

Parameters:

Parameter	Description
mInst	The controller instance.
percent	A double specifying the decimal percentage of spindle override.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Set the spindle override to 50% (.5 == 50%)
MINSTANCE mInst = 0;
int rc = mcSpindleSetOverride(mInst, 0.5);
```

mcSpindleSetRange

C/C++ Syntax:

```
int mcSpindleSetRange(
  MINSTANCE mInst,
  int Range);
```

LUA Syntax:

```
rc = mc.mcSpindleSetRange(
  number mInst,
  number Range)
```

Description: Set the current spindle range.

Parameters:

Parameter	Description
mInst	The controller instance.
Range	An integer specifying the spindle range.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_SPIN_RANGE_NOT_FOUND	**Range** specified an invalid spindle range.

Notes:
None.

Usage:

```
// Set the current spindle range to the first range (0).
MINSTANCE mInst = 0;
mcSpindleSetRange(mInst, 0);
```

mcSpindleSetReverse

C/C++ Syntax:

```
int mcSpindleSetReverse(
  MINSTANCE mInst,
  int Range,
  BOOL Reversed);
```

LUA Syntax:

```
rc = mc.mcSpindleSetReverse(
  number mInst,
  number Range,
  number Reversed)
```

Description: Set the spindle as being reversed for the given spindle range. e.g back gear.

Parameters:

Parameter	Description
mInst	The controller instance.
Range	An integer specifying the spindle range.
Reversed	A BOOL specifying whether the spindle is reversed for the given range.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_SPIN_RANGE_NOT_FOUND	**Range** specified an invalid spindle range.

Notes:
None.

Usage:

```
// Set spindle range 1 as being reversed.
MINSTANCE mInst = 0;
```

```
int rc = mcSpindleSetReverse(mInst, 1, TRUE);
```

mcSpindleSetSensorRPM

C/C++ Syntax:

```
int mcSpindleSetSensorRPM(
  MINSTANCE mInst,
  double RPM);
```

LUA Syntax:

```
rc = mc.mcSpindleSetSensorRPM(
  number mInst,
  number RPM)
```

Description: Set the spindle sensor RPM.

Parameters:

Parameter	Description
mInst	The controller instance.
RPM	A double specifying the spindle sensor RPM.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
This API function is what spindle control plugins should use to report the spindle speed.

Usage:

```
// Set the spindle sensor RPM to 1000
MINSTANCE mInst = 0;
int rc = mcSpindleSetSensorRPM(mInst, 1000);
```

mcSpindleSetSpeedCheckEnable

C/C++ Syntax:

```
int mcSpindleSetSpeedCheckEnable(
  MINSTANCE mInst,
  BOOL enable);
```

LUA Syntax:

```
rc = mc.mcSpindleSetSpeedCheckEnable(
  number mInst,
  number enable)
```

Description: Enable or disable the spindle speed check before feed moves.

Parameters:

Parameter	Description
mInst	The controller instance.
enable	A BOOL specifying the state of spindle speed check.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Spindle speed check is used to make sure the spindle is at an acceptable speed before a feed move.

See also: mcSpindleSetSpeedCheckPercent() and mcSpindleGetSpeedCheckPercent().

Usage:

```
// Enable spindle speed checking.
MINSTANCE mInst = 0;
int rc = mcSpindleSetSpeedCheckEnable(mInst, TRUE);
```

mcSpindleSetSpeedCheckPercent

C/C++ Syntax:

```
int mcSpindleSetSpeedCheckPercent(
  MINSTANCE mInst,
  double percent);
```

LUA Syntax:

```
rc = mc.mcSpindleSetSpeedCheckPercent(
  number mInst,
  number percent)
```

Description: Set the allowable percentage tha the spindle speed can fluctuate.

Parameters:

Parameter	Description
mInst	The controller instance.
percent	A double specifying the percentage.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Set the speed check percentage to 5%.  100 == 100%
MINSTANCE mInst = 0;
int rc = mcSpindleSetSpeedCheckPercent(mInst, 5);
```

mcSpindleSpeedCheck

C/C++ Syntax:

```
int mcSpindleSpeedCheck(
  MINSTANCE mInst,
  int *SpeedOk);
```

LUA Syntax:

```
SpeedOk, rc = mc.mcSpindleSpeedCheck(
  number mInst)
```

Description: Check if the spindle speed is withing the allowable range.

Parameters:

Parameter	Description
mInst	The controller instance.
SpeedOk	The address of a BOOL to receive the spindle speed check status.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_NOT_NOW	The spindle is currently off.

Notes:
None.

Usage:

```
// Check to see if the spidnle speed is in the allowed range.
MINSTANCE mInst = 0;
BOOL speedOk;
int rc = mcSpindleSpeedCheck(mInst, &speedOk;);
```

mcToolGetCurrent

C/C++ Syntax:

```
int mcToolGetCurrent(
  MINSTANCE mInst,
  int *toolnum);
```

LUA Syntax:

```
toolnum, rc = mc.mcToolGetCurrent(
  number mInst)
```

Description: Retrieve the current tool number.

Parameters:

Parameter	Description
mInst	The controller instance.
toolnum	The address of an integer to receive the current tool number.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Tool numbers start at 1 (base 1).

Usage:

```
// Get the current tool.
MINSTANCE mInst = 0;
int toolnum = 0;
int rc = mcToolGetCurrent(mInst, &toolnum;);
```

mcToolGetData

C/C++ Syntax:

```
int mcToolGetData(
  MINSTANCE mInst,
  int type,
  int toolNumber,
  double *value);
```

LUA Syntax:

```
value, rc = mc.mcToolGetData(
  number mInst,
  number type,
  number toolNumber)
```

Description: Retrieve the offset values for mill and turn tools.

Parameters:

Parameter	Description
mInst	The controller instance.
type	MTOOL_MILL_X, MTOOL_MILL_X_W, MTOOL_MILL_Y, MTOOL_MILL_Y_W, MTOOL_MILL_HEIGHT, MTOOL_MILL_HEIGHT_W, MTOOL_MILL_RAD, MTOOL_MILL_RAD_W, MTOOL_MILL_POCKET, MTOOL_LATHE_X, MTOOL_LATHE_X_W, MTOOL_LATHE_Y, MTOOL_LATHE_Y_W, MTOOL_LATHE_Z, MTOOL_LATHE_Z_W, MTOOL_LATHE_POCKET, MTOOL_LATHE_TIPRAD, MTOOL_MILL_RAD, MTOOL_LATHE_TIPDIR, MTOOL_MILL_RAD_W, MTOOL_LATHE_TOOLSLIDE, MTOOL_TYPE
toolNumber	Specifies the tool number.
value	The address of a double to receive the value for the specified offset type and the specified tool.

Returns:

Return Code	Description

MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**toolNumber** is out of range.

Notes:
Used to get the offset of a tool and can be used at anytime.

Usage:

```
MINSTANCE mInst = 0;
int toolnum = 5;
double val = 0;
// Get the tool height offset for tool number 5.
int rc = mcToolGetData(mInst, MTOOL_MILL_HEIGHT, toolnum, &val;);
```

mcToolGetDesc

C/C++ Syntax:

```
int mcToolGetDesc(
  MINSTANCE mInst,
  int toolnum,
  char *buff,
  size_t bufsize);
```

LUA Syntax:

```
buff, rc = mc.mcToolGetDesc(
  number mInst,
  number toolnum)
```

Description: Get the text string to describe the tool

Parameters:

Parameter	Description
mInst	The controller instance.
toolnum	An integer specifying the tool number for which to get the description.
buff	The address of a string buffer to receive the tool description.
bufsize	The length of the string buffer.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**toolnum** is out of range.

Notes:
Used to get the tool description for the tool table or run screen.

Usage:

```
MINSTANCE mInst = 0;
char desc[128];
```

```
int toolnum = 5;
// Get the description of tool number 5 for controller 0.
int rc = mcToolGetDesc(mInst, toolnum, desc, sizeof(desc));
```

mcToolGetSelected

C/C++ Syntax:

```
int mcToolGetSelected(
  MINSTANCE mInst,
  int *toolnum);
```

LUA Syntax:

```
toolnum, rc = mc.mcToolGetSelected(
  number mInst)
```

Description: Retrieve the selected (next) tool number.

Parameters:

Parameter	Description
mInst	The controller instance.
toolnum	The address of an integer to receive the selected tool number.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Get the selected tool number.
MINSTANCE mInst = 0;
int toolnum = 0;
int rc = mcToolGetSelected(mInst, &toolnum;);
```

mcToolLoadFile

C/C++ Syntax:

```
int mcToolLoadFile(
  MINSTANCE mInst,
  const char *FileToLoad);
```

LUA Syntax:

```
rc = mc.mcToolLoadFile(
  number mInst,
  string FileToLoad)
```

Description: Load a tool table into the core.

Parameters:

Parameter	Description
mInst	The controller instance.
FileToLoad	A string buffer specifying the tool table file to load.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_FILE_EMPTY	The specified tool table file was empty.

Notes:
Used to load a tool table file on a per profile basis.

Usage:

```
// Load a tool table.
MINSTANCE mInst = 0;
int rc = mcToolLoadFile(mInst, "tooltable.tls");
```

mcToolPathCreate

C/C++ Syntax:

```
int mcToolPathCreate(
  MINSTANCE mInst,
  void *window);
```

LUA Syntax:

```
N/A
```

Description: Create an OpenGL window to display the tool path.

Parameters:

Parameter	Description
mInst	The controller instance.
window	The parent window handle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
This API function is specific to GUI programming.
Passing the window handle is all that is needed to draw the toolpath.
For the Windows Platform, **window** should be the HWND of the parent window. For Linux and Mac, the wxWidgets wxWindow handle must be used.

Usage:

```
wxPanel* m_tpTP;

MINSTANCE mInst = 0;
m_tpTP = new wxPanel(itemPanel129, ID_TOOLPATH_PANEL, wxDefaultPosition, wxDefau
rc = m_tpTP->Show();
// Pass the handle of the tool path.
mcToolPathCreate(mInst, m_tpTP->GetHWND());
```

mcToolPathDelete

C/C++ Syntax:

```
int mcToolPathDelete(
  MINSTANCE mInst,
  void *parent);
```

LUA Syntax:

```
N/A
```

Description:

Parameters:

Parameter	Description
mInst	The controller instance.
parent	The handle of the tool paths parent window.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

This API function is specific to GUI programming.
For the Windows Platform, **window** should be the HWND of the parent window. For Linux and Mac, the wxWidgets wxWindow handle must be used.

Usage:

```
// Delete an existing tool path.
MINSTANCE mInst = 0;
HWND parent; // A previously valid window handle.
int rc = mcToolPathDelete(mInst, parent);
```

mcToolPathGenerate

C/C++ Syntax:

```
int mcToolPathGenerate(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcToolPathGenerate(
  number mInst)
```

Description: Generates the toolpath to refresh any changes that may have been done (like cutter comp or fixure offset changes).

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Generation of the tool path should only be done with the Gocd file is not running.

Usage:

```
MINSTANCE mInst = 0;
// Regenerate the toolpaths for controller instance 0
int rc = mcToolPathGenerate(mInst);
```

mcToolPathGenerateAbort

C/C++ Syntax:

```
int mcToolPathGenerateAbort(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcToolPathGenerateAbort(
  number mInst);
```

Description: Abort tool path generation.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Used if there is an endless loop in the Gcode or if a file regen needs to be stopped.

Usage:

```
MINSTANCE mInst = 0;
// Abort the regeneration of the toolpath for controller 0.
int rc = mcToolPathGenerateAbort(mInst);
```

mcToolPathGeneratedPercent

C/C++ Syntax:

```
int mcToolPathGeneratedPercent(
  MINSTANCE mInst,
  double *percent);
```

LUA Syntax:

```
percent, rc = mc.mcToolPathGeneratedPercent(
  number mInst)
```

Description: Retrieve the tool path generation percentage completed.

Parameters:

Parameter	Description
mInst	The controller instance.
percent	The address of a double to receive the tool path completion percentage. (0 to 100)

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
This function is to be used to show the path generation progress in a progress dialog when a regen has been initiated.

Usage:

```
MINSTANCE mInst = 0;
double pdone = 0;

while (pdone != 100) {
 // Get the percent of the file that is loaded.
 mcToolPathGeneratedPercent(mInst, &pdone;);
 // Post the pdone to some dialog.
 Sleep(250);
}
```

mcToolPathGetAAxisPosition

C/C++ Syntax:

```
mcToolPathGetAAxisPosition(MINSTANCE mInst, double *xPos, double *yPos, double *
```

LUA Syntax:

```
mcToolPathGetAAxisPosition(MINSTANCE mInst, double *xPos, double *yPos, double *
```

Description:

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

Usage:

```
//
MINSTANCE mInst = 0;
mcToolPathGetAAxisPosition(MINSTANCE mInst, double *xPos, double *yPos, double *
```

mcToolPathGetARotationAxis

C/C++ Syntax:

```
mcToolPathGetARotationAxis(MINSTANCE mInst, int *axis);
```

LUA Syntax:

```
mcToolPathGetARotationAxis(MINSTANCE mInst, int *axis);
```

Description:

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

Usage:

```
//
MINSTANCE mInst = 0;
mcToolPathGetARotationAxis(MINSTANCE mInst, int *axis);
```

mcToolPathGetAxisColor

C/C++ Syntax:

```
int mcToolPathGetAxisColor(
  MINSTANCE mInst,
  unsigned long *axiscolor,
  unsigned long *limitcolor);
```

LUA Syntax:

```
axiscolor, limitcolor, rc = mc.mcToolPathGetAxisColor(
  number mInst)
```

Description: Retrieve the current axis and limit tool path colors.

Parameters:

Parameter	Description
mInst	The controller instance.
axiscolor	The address of an unsigned long to receive the axis color.
limitcolor	The address of an unsigned long to receive the sof limit color.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
The colors are in RGB format converted into unsigned longs.

Usage:

```
// Get the axis and limit colors.
MINSTANCE mInst = 0;
int rc = mcToolPathGetAxisColor(MINSTANCE mInst, unsigned long *axiscolor, unsig
```

mcToolPathGetBackColor

C/C++ Syntax:

```
int mcToolPathGetBackColor(
  MINSTANCE mInst,
  unsigned long *topcolor,
  unsigned long *botcolor);
```

LUA Syntax:

```
topcolor, botcolor, rc = mc.mcToolPathGetBackColor(
  number mInst)
```

Description: Get the background top and bottom colors.

Parameters:

Parameter	Description
mInst	The controller instance.
topcolor	The address of an unsigned long that receives the top color of the background.
botcolor	The address of an unsigned long that receives the bottom color of the background.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
The colors are in RGB format converted into unsigned longs.

Usage:

```
MINSTANCE mInst = 0;
unsigned long topcol, botcol;

// Get the tool background top and bottom colors
int rc = mcToolPathGetBackColor(m_inst, &topcol;, &botcol;);
```

mcToolPathGetDrawLimits

C/C++ Syntax:

```
int mcToolPathGetDrawLimits(
  MINSTANCE mInst,
  BOOL *drawlimits);
```

LUA Syntax:

```
drawlimits, rc = mcToolPathGetDrawLimits(
  number mInst)
```

Description: Retrieve the status of the soft limit display.

Parameters:

Parameter	Description
mInst	The controller instance.
drawlimits	The address of a BOOL to receives the state of soft limit display.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
MINSTANCE mInst = 0;
BOOL drawlimits = FALSE;

// See if the soft limits bounding box is beeing shown in the toolpath.
int rc = mcToolPathGetDrawLimits(mInst, &drawlimits;);
```

mcToolPathGetExecution

C/C++ Syntax:

```
int mcToolPathGetExecution(
  MINSTANCE mInst,
  unsigned long exNum,
  void **data,
  unsigned long *len);
```

LUA Syntax:

```
N/A
```

Description:

Parameters:

Parameter	Description
mInst	The controller instance.
exNum	An unsigned long specifying the execution number.
data	The address of a void pointer to receive the move execution info structure.
len	The address of an unsigned long the receive the length of the move execution infomation.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

This structure is not in the MachAPI.h header file. And it is subject to change without notice. Any new items will be added onto the end of the structure thus it is important to use this API call correctly and not rely on the length of the structure given below.

```
struct Executions
{
 bool linear : 1; //Is this a linear move?
 bool rapid : 1;  //Is it a Rapid move?
 bool inc : 1;  //Is this an inc move?
 bool comp : 1;
```

```
 bool rotation : 1;      //arc direction
 bool UsedAxis0 : 1;
 bool UsedAxis1 : 1;
 bool UsedAxis2 : 1;
 bool UsedAxis3 : 1;
 bool UsedAxis4 : 1;
 bool UsedAxis5 : 1;
 bool UsedAxis6 : 1;
 bool UsedAxis7 : 1;
 bool UsedAxis8 : 1;
 bool UsedAxis9 : 1;
 bool UsedAxis10 : 1;
 float end[6];
 float center[3];
 float normal[3];
 unsigned int LineNumber; // Line number in the file...
 unsigned int ExecutionID; // Number of the move from the Gcode interperter
 float FeedRate;   // Feedrate of the move
 unsigned char ToolNumber; // Tool number of the move used to show offsets..
 unsigned char Fixture;  // The fixture.
};
```

Usage:

```
//
MINSTANCE mInst = 0;
unsigned long exNum = 0;
unsigned long len;
void *data;
int rc;
struct Executions *ex;
// Get the length of the move execution structure first.
while (mcToolPathGetExecution(mInst, exNum, NULL, &len;) == MERROR_NOERROR) {
 // Allocate some mem to receive the data.
 data = malloc(len);
 if (data != NULL) {
  // Get the data.
  if (mcToolPathGetExecution(mInst, exNum, &data;, &len;) == MERROR_NOERROR) {
   // cast the void pointer to the current structure.
   ex = (struct Executions *)data;
   ...
   free(data);
  }
 }
}
```

mcToolPathGetFollowMode

C/C++ Syntax:

```
int mcToolPathGetFollowMode(
  MINSTANCE mInst,
  BOOL *enabled);
```

LUA Syntax:

```
enabled, rc = mc.mcToolPathGetFollowMode(
  number mInst)
```

Description: Retrieve the status of the tool path jog follow mode.

Parameters:

Parameter	Description
mInst	The controller instance.
enabled	The address of a BOOL to receive the jog follow mode state.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// See if jog follow is enabled.
MINSTANCE mInst = 0;
BOOL enabled = FALSE;
int rc = mcToolPathGetFollowMode(mInst, &enabled;);
```

mcToolPathGetGenerating

C/C++ Syntax:

```
int mcToolPathGetGenerating(
  MINSTANCE mInst,
  int *pathGenerating);
```

LUA Syntax:

```
pathGenerating, rc = mc.mcToolPathGetGenerating(
  number mInst)
```

Description: Retrieve the state of the tool path generation.

Parameters:

Parameter	Description
mInst	The controller instance.
pathGenerating	The address A pointer to an int to receive the path generation state (MC_ON, MC_OFF).

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**pathGenerating** is NULL.

Notes:
This function is used after a file load, or regen to keep the user from using any controles as the machine is loading.

Usage:

```
MINSTANCE mInst = 0;
BOOL IsGen = FALSE;
// See if the tool path is generating.
int rc = mcGetPathGenerating(mInst, &IsGen;);
```

mcToolPathGetLeftMouseDn

C/C++ Syntax:

```
int mcToolPathGetLeftMouseDn(
  MINSTANCE mInst,
  double *x,
  double *y,
  double *z);
```

LUA Syntax:

```
x, y, z, rc = mc.mcToolPathGetLeftMouseDn(
  number mInst)
```

Description: Retrieve the xyz coordinates of the mouse in the tool path when the left button goes down.

Parameters:

Parameter	Description
mInst	The controller instance.
x	The address to a double to revceive the x coordinate.
y	The address to a double to revceive the y coordinate.
z	The address to a double to revceive the z coordinate.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Get the mouse coordinates in the tool path.
MINSTANCE mInst = 0;
double x, y, z;
int rc = mcToolPathGetLeftMouseDn(MINSTANCE mInst, double *x, double *y, double
```

mcToolPathGetLeftMouseUp

C/C++ Syntax:

```
int mcToolPathGetLeftMouseUp(
  MINSTANCE mInst,
  double *x,
  double *y,
  double *z);
```

LUA Syntax:

```
x, y, z, rc = mc.mcToolPathGetLeftMouseUp(
  number mInst)
```

Description: Retrieve the xyz coordinates of the mouse in the tool path when the left button comes up.

Parameters:

Parameter	Description
mInst	The controller instance.
x	The address to a double to revceive the x coordinate.
y	The address to a double to revceive the y coordinate.
z	The address to a double to revceive the z coordinate.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Get the mouse coordinates in the tool path.
MINSTANCE mInst = 0;
double x, y, z;
int rc = mcToolPathGetLeftMouseUp(MINSTANCE mInst, double *x, double *y, double
```

mcToolPathGetPathColor

C/C++ Syntax:

```
int mcToolPathGetPathColor(
  MINSTANCE mInst,
  unsigned long *rapidcolor,
  unsigned long *linecolor,
  unsigned long *arccolor,
  unsigned long *highlightcolor);
```

LUA Syntax:

```
rapidcolor, linecolor, arccolor, highlightcolor, rc = mcToolPathGetPathColor(
  number mInst)
```

Description: Retrieve the tool path display colors.

Parameters:

Parameter	Description
mInst	The controller instance.
rapidcolor	The address of an unsigned long that receives the rapid (G00) color of the toolpath.
linecolor	The address of an unsigned long that receives the line (G01) color of the toolpath.
arccolor	The address of an unsigned long that receives the arc (G02/G03) color of the toolpath.
highlightcolor	The address of an unsigned long that receives the past moves color of the toolpath.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
The colors are in RGB format converted into unsigned longs.

Usage:

```
MINSTANCE mInst = 0;
unsigned long rapcol, lincol, arccol, highcol;
// Get the tool path colors
int rc = mcToolPathGetPathColor(m_inst, &rapcol;, &lincol;, &arccol;, &highcol;)
```

mcToolPathIsSignalMouseClicks

C/C++ Syntax:

```
int mcToolPathIsSignalMouseClicks(
  MINSTANCE mInst,
  BOOL *enabled);
```

LUA Syntax:

```
enabled, rc = mcToolPathIsSignalMouseClicks(
  number mInst)
```

Description: Determine if tool path mouse click events are signaled.

Parameters:

Parameter	Description
mInst	The controller instance.
enabled	The address of a BOOL to receive the state of the tool path mouse clicks.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

Usage:

```
// See if we are signaling mouse click events in the tool path.
MINSTANCE mInst = 0;
BOOL enabled = FALSE;
int rc = mcToolPathIsSignalMouseClicks(mInst, &enabled;);
```

mcToolPathSetAAxisPosition

C/C++ Syntax:

```
mcToolPathSetAAxisPosition(MINSTANCE mInst, double xPos, double yPos, double zPos
```

LUA Syntax:

```
mcToolPathSetAAxisPosition(MINSTANCE mInst, double xPos, double yPos, double zPos
```

Description:

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

Usage:

```
//
MINSTANCE mInst = 0;
mcToolPathSetAAxisPosition(MINSTANCE mInst, double xPos, double yPos, double zPos
```

mcToolPathSetARotationAxis

C/C++ Syntax:

```
mcToolPathSetARotationAxis(MINSTANCE mInst, int axis);
```

LUA Syntax:

```
mcToolPathSetARotationAxis(MINSTANCE mInst, int axis);
```

Description:

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

Usage:

```
//
MINSTANCE mInst = 0;
mcToolPathSetARotationAxis(MINSTANCE mInst, int axis);
```

mcToolPathSetAxisColor

C/C++ Syntax:

```
int mcToolPathSetAxisColor(
  MINSTANCE mInst,
  unsigned long axiscolor,
  unsigned long limitcolor);
```

LUA Syntax:

```
rc = mc.mcToolPathSetAxisColor(
  number mInst,
  number axiscolor,
  number limitcolor)
```

Description: Set the tool path axis and limit colors.

Parameters:

Parameter	Description
mInst	The controller instance.
axiscolor	The color of the top of the axis lines in the toolpath.
limitcolor	The color of the softlimits color bounding box in the toolpath.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
The colors are in RGB format converted into unsigned long.

Usage:

```
MINSTANCE mInst = 0;
unsigned long axiscolor;
unsigned long limitcolor;

//red
axiscolor =  (255<<0);//red
axiscolor += (0<<8);//green
```

```
axiscolor += (0<<16);//blue

//Yellow
limitcolor =  (255<<0);//red
limitcolor += (255<<8);//green
limitcolor += (128<<16);//blue

// Set the axis and limit colors
int rc = mcToolPathSetAxisColor(mInst, axiscolor, limitcolor);
```

mcToolPathSetBackColor

C/C++ Syntax:

```
int mcToolPathSetBackColor(
  MINSTANCE mInst,
  unsigned long topcolor,
  unsigned long botcolor);
```

LUA Syntax:

```
rc = mc.mcToolPathSetBackColor(
  number mInst,
  number topcolor,
  number botcolor)
```

Description: Set the tool path background color.

Parameters:

Parameter	Description
mInst	The controller instance.
topcolor	The color of the top of the background.
botcolor	The color of the bottom of the background.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Used to set the top and botom background color of toolpath. The colors are in RGB format converted into unsigned long.

Usage:

```
MINSTANCE mInst = 0;
unsigned long topcolor;
unsigned long botcolor

//Dark Blue Top
topcolor = (28<<0);
topcolor += (28<<8);
```

```
topcolor += (213<<16);
//Light Blue Bottom
botcolor = (164<<0);
botcolor += (156<<8);
botcolor += (228<<16);

// Set the background color
int rc = mcToolPathSetBackColor(mInst, topcolor, botcolor);
```

mcToolPathSetDrawLimits

C/C++ Syntax:

```
int mcToolPathSetDrawLimits(
  MINSTANCE mInst,
  BOOL drawlimits);
```

LUA Syntax:

```
rc = mc.mcToolPathSetDrawLimits(
  number mInst,
  number drawlimits)
```

Description: Enable or disable the drawing of soft limits on the tool paths.

Parameters: (drawlimits, A BOOL specifying whether soft limits are drawn on the tool path.

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Turn on soft limits in the tool path.
MINSTANCE mInst = 0;
int rc = mcToolPathSetDrawLimits(mInst, TRUE);
```

mcToolPathSetFollowMode

C/C++ Syntax:

```
int mcToolPathSetFollowMode(
  MINSTANCE mInst,
  BOOL enabled);
```

LUA Syntax:

```
rc = mc.mcToolPathSetFollowMode(
  number mInst,
  number enabled)
```

Description: Enable or disable tool path jog follow mode.

Parameters:

Parameter	Description
mInst	The controller instance.
enabled	A BOOL specifying the state of the tool path jog follow mode.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
None.

Usage:

```
// Turn on tool path jog following.
MINSTANCE mInst = 0;
int rc = mcToolPathSetFollowMode(mInst, TRUE);
```

mcToolPathSetPathColor

C/C++ Syntax:

```
int mcToolPathSetPathColor(
  MINSTANCE mInst,
  unsigned long rapidcolor,
  unsigned long linecolor,
  unsigned long arccolor,
  unsigned long highlightcolor);
```

LUA Syntax:

```
rc = mc.mcToolPathSetPathColor(
  number mInst,
  number rapidcolor,
  number linecolor,
  number arccolor,
  number highlightcolor)
```

Description: Set the tool path line display colors.

Parameters:

Parameter	Description
mInst	The controller instance.
rapidcolor	The color of the rapid (G00) moves in the toolpath.
linecolor	The color of the linear feed (G01) moves in the toolpath.
arccolor	The color of the arc (G03/G02) moves in the toolpath.
highlightcolor	The color of the moves that have been done in the toolpath.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
The colors are in RGB format converted into unsigned longs.

Usage:

```
// Set the tool path colors
MINSTANCE mInst = 0;
unsigned long rapidcolor;
unsigned long linecolor;
unsigned long arccolor;
unsigned long highlightcolor;

rapidcolor =  (254<<0);//Red
rapidcolor += (0<<8);//Green
rapidcolor += (0<<16);//Blue

linecolor =  (0<<0);//Red
linecolor += (254<<8);//Green
linecolor += (0<<16);//Blue

arccolor =  (0<<0);//Red
arccolor += (254<<8);//Green
arccolor += (0<<16);//Blue

highlightcolor =  (254<<0);//Red
highlightcolor += (254<<8);//Green
highlightcolor += (254<<16);//Blue

int rc = mcToolPathSetPathColor(mInst, rapidcolor, linecolor, arccolor, highlight
```

mcToolPathSetSignalMouseClicks

C/C++ Syntax:

```
int mcToolPathSetSignalMouseClicks(
  MINSTANCE mInst,
  BOOL enabled);
```

LUA Syntax:

```
rc = mc.mcToolPathSetSignalMouseClicks(
  number mInst,
  number enabled)
```

Description: Enable or Disable the tool path mouse click events.

Parameters:

Parameter	Description
mInst	The controller instance.
enabled	A BOOL specifying the state of the tool path mouse click event signaling.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:

Usage:

```
// Turn on mouse click events in the tool path.
MINSTANCE mInst = 0;
int rc = mcToolPathSetSignalMouseClicks(mInst, TRUE);
```

mcToolPathSetView

C/C++ Syntax:

```
int mcToolPathSetView(
  MINSTANCE mInst,
  void *parent,
  int view);
```

LUA Syntax:

```
N/A
```

Description: Set the tool path view orientation.

Parameters:

Parameter	Description
mInst	The controller instance.
parent	The tool path parent window handle.
view	An integer specifying the tool path view orientation. (MC_TPVIEW_TOP, MC_TPVIEW_BOTTOM, MC_TPVIEW_LEFT, MC_TPVIEW_RIGHT, or MC_TPVIEW_ISO)

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
This API function is specific to GUI programming.
For the Windows Platform, **window** should be the HWND of the parent window. For Linux and Mac, the wxWidgets wxWindow handle must be used.

Usage:

```
// Set the tool path orientation to the ISO view.
MINSTANCE mInst = 0;
HWND parent;  // A valid window handle.
int rc = mcToolPathSetView(mInst, parent, MCTPVIEW_ISO);
```

mcToolPathSetZoom

C/C++ Syntax:

```
int mcToolPathSetZoom(
  MINSTANCE mInst,
  void *parent,
  double zoom);
```

LUA Syntax:

```
N/A
```

Description: Set the zoom of the given tool path

Parameters:

Parameter	Description
mInst	The controller instance.
parent	The tool path parent window handle.
zoom	A double specifying the zoom percentage as a decimal.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
This API function is specific to GUI programming.
For the Windows Platform, **window** should be the HWND of the parent window. For Linux and Mac, the wxWidgets wxWindow handle must be used.
zoom should be a decimal percentage. e.g. 1.5 == 150%.

Usage:

```
// Set the toop path zoom percentage to 150%
MINSTANCE mInst = 0;
HWND parent;  // A valid window handle.
int rc = mcToolPathSetZoom(mInst, parent, 1.5);
```

mcToolPathUpdate

C/C++ Syntax:

```
int mcToolPathUpdate(
  MINSTANCE mInst,
  void *parent);
```

LUA Syntax:

```
N/A
```

Description: Redraw/update the OpenGL tool path.

Parameters:

Parameter	Description
mInst	The controller instance.
parent	The tool path parent window handle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
No Error.	
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
The mInst parameter was out of range.	

Notes:
Updates need to be run at every .1 sec or faseter to have a smooth tool path.
This API function is specific to GUI programming.
For the Windows Platform, **window** should be the HWND of the parent window. For Linux and Mac, the wxWidgets wxWindow handle must be used.

Usage:

```
MINSTANCE mInst = 0;
HWND parent;  // A valid window handle.
int mcToolPathUpdate(mInst, parent);
```

mcToolSaveFile

C/C++ Syntax:

```
int mcToolSaveFile(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc =  mcToolSaveFile(
  number mInst)
```

Description: Save a previously loaded tool table file.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_FILE_INVALID	No tool table fie was previously loaded.

Notes:
None.

Usage:

```
// Save the previously loaded tool table file.
MINSTANCE mInst = 0;
int rc = mcToolSaveFile(mInst);
```

mcToolSetCurrent

C/C++ Syntax:

```
int mcToolSetCurrent(
  MINSTANCE mInst,
  int toolnum);
```

LUA Syntax:

```
rc = mc.mcToolSetCurrent(
  number mInst,
  number toolnum)
```

Description: Set the current tool to the specified tool.

Parameters:

Parameter	Description
mInst	The controller instance.
toolnum	An integer specifying the tool.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**toolnum** is out of range.

Notes:
None.

Usage:

```
// Set the current tool to tool #1.
MINSTANCE mInst = 0;
int rc = mcToolSetCurrent(mInst, 1);
```

mcToolSetData

C/C++ Syntax:

```
int mcToolSetData(
  MINSTANCE mInst,
  int Type,
  int Toolnumber,
  double val);
```

LUA Syntax:

```
rc = mc.mcToolSetData(
  number mInst,
  number Type,
  number Toolnumber,
  number val);
```

Description: Set the offset values for each tool.

Parameters:

Parameter	Description
mInst	The controller instance.
Type	MTOOL_MILL_X, MTOOL_MILL_X_W, MTOOL_MILL_Y, MTOOL_MILL_Y_W, MTOOL_MILL_HEIGHT, MTOOL_MILL_HEIGHT_W, MTOOL_MILL_RAD, MTOOL_MILL_RAD_W, MTOOL_MILL_POCKET, MTOOL_LATHE_X, MTOOL_LATHE_X_W, MTOOL_LATHE_Y, MTOOL_LATHE_Y_W, MTOOL_LATHE_Z, MTOOL_LATHE_Z_W, MTOOL_LATHE_POCKET, MTOOL_LATHE_TIPRAD, MTOOL_MILL_RAD, MTOOL_LATHE_TIPDIR
Toolnumber	Tool number to be set
val	Value to set the selected offset.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.

MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**Toolnumber** is out of range.

Notes:
Used to set the offest of a tool and shouldn't be changed as Gcode is running.

Usage:

```
MINSTANCE mInst = 0;
int toolnum = 5;
double val = 0;
// Set the tool height wear offset to zero.
int rc = mcToolSetData(mInst, MTOOL_MILL_HEIGHT_W, toolnum, val);
```

mcToolSetDesc

C/C++ Syntax:

```
int mcToolSetDesc(
  MINSTANCE mInst,
  int toolnum,
  const char *tdsc);
```

LUA Syntax:

```
rc = mc.mcToolSetDesc(
  number mInst,
  number toolnum,
  string tdsc);
```

Description: Set the description for the specified tool in the tool table.

Parameters:

Parameter	Description
mInst	The controller instance.
toolnum	Tool number to which to set the description.
tdsc	A string buffer that specifies the description of the tool.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.
MERROR_INVALID_ARG	**toolnumber** is out of range.

Notes:
Used to set the description of a tool for the tool table or it can be used to allow the user to change it on the program run page.

Usage:

```
MINSTANCE mInst = 0;
char desc[128] = "My Best 1/2 inch endmill";
int toolnum = 5;
// Set the description of tool number 5.
int rc = mcToolSetDesc(mInst, toolnum, desc);
```

mcCntlCloseGCodeFile

C/C++ Syntax:

```
int mcCntlCloseGCodeFile(
  MINSTANCE mInst);
```

LUA Syntax:

```
rc = mc.mcCntlCloseGCodeFile(
  number mInst)
```

Description: Close Gocde file.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The **mInst** parameter was out of range.

Notes:
Used to close a Gcode file if the Gcode File needs to be edited

Usage:

```
int mInst=0;
mcCntlCloseGCodeFile(mInst); // Close the Gcode file in controller 0.
```